



JoramMQ 1.18

JMS Reference Manual

last modification: Sep 26, 2023

Contents

1. Introduction.....	13
2. Class Hierarchy Index.....	15
3. Class List.....	17
4. Unified JMS API Class reference.....	19
4.1. ConnectionFactory.....	19
4.2. LocalConnectionFactory.....	24
4.3. TcpConnectionFactory.....	24
4.4. FactoryParameters.....	25
4.5. Connection.....	28
4.6. ConnectionMetaData.....	34
4.7. Session.....	36
4.8. MessageConsumer.....	49
4.9. MessageProducer.....	52
4.10. QueueBrowser.....	56
4.11. Destination.....	57
4.12. Queue.....	65
4.13. TemporaryQueue.....	73
4.14. Topic.....	74
4.15. TemporaryTopic.....	78
4.16. User.....	79
4.17. Subscription.....	87
4.18. ClusterConnectionFactory.....	88
4.19. ClusterDestination.....	89

4.20. ClusterQueue.....	91
4.21. ClusterTopic.....	91
4.22. DeadMQueue.....	92
4.23. XALocalConnectionFactory.....	93
4.24. XATcpConnectionFactory.....	94
4.25. XAConnection.....	94
4.26. XASession.....	95
4.27. PooledConnectionFactory.....	97
4.28. PooledConnection.....	98
4.29. ConnectionPool.....	100
5. Messages API Class Reference.....	102
5.1. Message.....	102
5.2. BytesMessage.....	114
5.3. MapMessage.....	121
5.4. StreamMessage.....	127
5.5. ObjectMessage.....	134
5.6. TextMessage.....	134
6. Destination API Class Reference.....	136
6.1. JMSAcquisitionQueue.....	136
6.2. JMSAcquisitionTopic.....	138
6.3. JMSDistributionQueue.....	140
6.4. JMSDistributionTopic.....	142
6.5. RestAcquisitionQueue.....	145
6.6. RestDistributionQueue.....	149
6.7. AMQPACquisitionQueue.....	154
6.8. AMQPACquisitionTopic.....	156
6.9. AMQPDistributionQueue.....	158

6.10. AMQPDistributionTopic.....	161
6.11. CollectorQueue.....	163
6.12. CollectorTopic.....	165
6.13. FtpQueue.....	167
6.14. MailAcquisitionQueue.....	169
6.15. MailAcquisitionTopic.....	171
6.16. MailDistributionQueue.....	173
6.17. MailDistributionTopic.....	175
6.18. MonitoringQueue.....	177
6.19. MonitoringTopic.....	179
6.20. SchedulerQueue.....	181
7. Message Interception.....	182
7.1. Client-side interceptor.....	182
7.1.1. Writing an interceptor.....	182
7.1.2. MessageInterceptor Interface.....	182
7.1.3. Configuring a ConnectionFactory with interceptors.....	183
7.2. Server-side interceptor.....	183
7.2.1. Writing an interceptor.....	184
7.2.2. MessageInterceptor Interface.....	184
7.2.3. Configuring interceptors on destination.....	184
7.2.4. Configuring interceptors on user's connections.....	185
8. PTP JMS API Class Reference.....	187
8.1. QueueLocalConnectionFactory.....	187
8.2. QueueTcpConnectionFactory.....	187
8.3. QueueConnection.....	188
8.4. QueueSession.....	189
8.5. QueueReceiver.....	190
8.6. QueueSender.....	191

8.7. XAQueueLocalConnectionFactory	192
8.8. XAQueueTcpConnectionFactory	192
8.9. XAQueueConnection	193
8.10. XAQueueSession	194
9. P/S JMS API Class Reference	196
9.1. TopicLocalConnectionFactory	196
9.2. TopicTcpConnectionFactory	196
9.3. TopicConnection	197
9.4. TopicSession	198
9.5. TopicPublisher	199
9.6. TopicSubscriber	201
9.7. XATopicLocalConnectionFactory	202
9.8. XATopicTcpConnectionFactory	202
9.9. XATopicConnection	203
9.10. XATopicSession	204
10. Application Server Class Reference	205
10.1. MultiSessionConsumer	205
11. Administration API Class Reference	206
11.1. AdminException	206
11.2. AdminHelper	206
11.3. AdminItf	209
11.4. JoramSaxWrapper	224
11.5. AdminModule	225
11.6. AdminWrapper	244
11.7. ZeroconfJoramServer	261
12. XML Administration Scripts	262

12.1. Introduction.....	262
12.2. Administration connection.....	262
12.2.1. LocalAdminModule.....	262
12.2.2. TcpAdminModule.....	262
12.2.3. SSLAdminModule.....	263
12.3. Naming.....	263
12.4. ConnectionFactory.....	263
12.4.1. LocalConnectionFactory.....	263
12.4.2. TcpConnectionFactory.....	264
12.4.3. SSLConnectionFactory.....	264
12.5. User.....	264
12.6. Destinations.....	265
12.6.1. Queue.....	265
12.6.2. Topic.....	266
12.7. JMS bridge destinations.....	266
12.7.1. JMSAcquisitionQueue.....	266
12.7.2. JMSDistributionQueue.....	266
12.7.3. JMSAcquisitionTopic.....	267
12.7.4. JMSDistributionTopic.....	267
12.7.5. JMSBridgeConnection.....	267
12.7.6. JMSConnectionService.....	268
12.8. Rest / JMS Bridge destinations.....	268
12.8.1. Rest / JMS Acquisition queue.....	268
12.8.2. Rest / JMS Distribution queue.....	269
12.9. AMQP bridge destinations.....	269
12.9.1. AMQPAcquisitionQueue.....	269
12.9.2. AMQPDistributionQueue.....	270
12.9.3. AMQPAcquisitionTopic.....	270
12.9.4. AMQPDistributionTopic.....	270
12.9.5. AMQPBridgeConnection.....	271
12.9.6. AMQPConnectionService.....	271

12.10. Specialized destinations.....	272
12.10.1. CollectorQueue.....	272
12.10.2. CollectorTopic.....	272
12.10.3. FtpQueue.....	272
12.10.4. MailAcquisitionQueue.....	272
12.10.5. MailAcquisitionTopic.....	273
12.10.6. MailDistributionQueue.....	273
12.10.7. MailDistributionTopic.....	273
12.10.8. MonitoringQueue.....	273
12.10.9. MonitoringTopic.....	273
12.10.10. SchedulerQueue.....	273
13. OSGi Configuration Class Reference.....	274
13.1. Introduction.....	274
13.2. JoramManagedService.....	276
13.3. Activator.....	276
13.4. ServiceConnectionFactory.....	277
13.5. ServiceUser.....	277
13.6. ServiceQueue.....	278
13.7. ServiceAcquisitionQueue.....	279
13.8. ServiceAliasQueue.....	280
13.9. ServiceDistributionQueue.....	281
13.10. ServiceSchedulerQueue.....	281
13.11. ServiceFtpQueue.....	282
13.12. ServiceTopic.....	283
13.13. ServiceAcquisitionTopic.....	284
13.14. ServiceDistributionTopic.....	285
13.15. ConnectionFactoryMSF.....	286
13.16. DestinationMSF.....	286
13.17. UserMSF.....	287

14. Shell Commands.....	289
14.1. A3 commands.....	289
14.2. MOM commands.....	289
14.3. JNDI commands.....	290
15. JMX API Class Reference.....	291
15.1. JoramAdmin.....	291
15.2. JoramAdminConnect.....	312
15.3. JoramAdminConnectMBean.....	313
15.4. JoramAdminMBean.....	314
15.5. LocalConnections.....	323
16. Joram-Spring.....	325
16.1. Introduction.....	325
16.2. The Joram server.....	327
16.2.1. JoramServerBeanDefinitionParser.....	327
16.2.2. JoramServer.....	328
16.3. The Joram administration wrapper.....	329
16.3.1. JoramAdminBeanDefinitionParser.....	329
16.3.2. JoramAdmin.....	330
16.4. The Joram connection factory.....	331
16.4.1. JoramLocalConnectionFactoryBeanDefinitionParser.....	331
16.4.2. JoramTcpConnectionFactoryBeanDefinitionParser.....	332
16.4.3. JoramConnectionFactory.....	332
16.5. The Joram destinations.....	333
16.5.1. JoramQueueBeanDefinitionParser.....	333
16.5.2. JoramTopicBeanDefinitionParser.....	334
16.5.3. JoramDestination.....	334
16.6. The Joram users.....	335
16.6.1. JoramUserBeanDefinitionParser.....	335

16.6.2. JoramUser.....	336
16.7. Tests configuration.....	337
16.7.1. JoramTestServerBeanDefinitionParser.....	337
16.8. Running Joram-spring example.....	337
16.8.1. The webapp descriptors (web.xml).....	338
16.8.2. The Spring context configuration descriptors.....	338
16.8.3. Running the example.....	340
17. Exceptions.....	341
17.1. JMS Exceptions.....	341
17.2. Administration Exceptions.....	341
18. JMX indicators.....	343
18.1. AgentServer.....	343
18.1.1. AgentServer.....	343
18.1.2. Engine.....	343
18.1.3. Agent.....	344
18.2. Transaction.....	345
18.2.1. NGTransaction.....	345
18.2.2. JDBCTransaction.....	346
18.3. JNDI.....	347
18.3.1. NamingContext.....	347
18.3.2. JNDI TCP connector.....	348
18.4. Joram#0.....	348
18.4.1. JMS Connection Manager.....	348
18.4.2. JMS TCP/SSL Connector.....	349
18.4.3. JMS Connection.....	349
18.4.4. Queue.....	349
18.4.5. Topic.....	351
18.4.6. User.....	351
18.4.7. JMSModule (bridge JMS).....	351

19. Environment variables.....	353
19.1. Server-side.....	353
19.1.1. Configuration.....	353
19.1.2. OSGi.....	355
19.1.3. Server Health checking.....	355
19.1.4. File transacted persistence.....	355
19.1.5. Database transacted persistence.....	357
19.1.6. Communication components.....	359
19.1.7. Joram service.....	364
19.1.8. JNDI Service.....	366
19.1.9. Tools.....	366
19.2. Client-side.....	366
19.2.1. ConnectionFactory.....	366
19.2.2. JMS MessageID.....	368
19.2.3. JNDI.....	368
20. Logging.....	370
20.1. Configuration.....	370
20.2. Topics.....	370
20.2.1. A3 – Agent runtime.....	370
20.2.2. Joram – JMS runtime.....	370
20.3. Error's messages server side.....	371
20.3.1. Initialisation / Starting.....	371
20.3.2. Running.....	371
20.3.3. Stopping.....	371
20.4. Error's messages client side.....	372
20.4.1. Administration.....	372
21. Deprecated List.....	373

Figures

Figure 4.1 - Inheritance diagram for Connection class.....	27
Figure 4.2 - Inheritance diagram for Session class.....	36
Figure 4.3 - Inheritance diagram for Destination class.....	56
Figure 4.4 - Inheritance diagram for Queue class.....	65
Figure 4.5 - Inheritance diagram for Topic class.....	73
Figure 5.1 - Inheritance diagram for Message class.....	101
Figure 11.1 - Inheritance diagram for AdminIf interface.....	208
Figure 11.2 - Inheritance diagram for AdminWrapper class.....	243
Figure 15.1 - Collaboration diagram for JoramAdmin class.....	290

1. Introduction

JORAM is a production ready distributed MOM (Message Oriented Middleware) already used within many critical operational applications, it provides a fully JMS compliant API (JMS 1.1, 2.0 and Jakarta/JMS 3.0). JORAM is developed on Linux and Windows, as the server and the client components are entirely written in Java, they are platform-neutral.

ARCHITECTURAL CONFIGURABILITY

JORAM greatly benefits from the new generation Message Oriented Middleware from ScalAgent D.T., an agent-based truly distributed architecture. The underlying innovative architecture allows distributed applications to be connected on a large-scale basis through Internet, enables load balancing and guarantees high availability and flexibility.

The main characteristic of JORAM is its distributed configurable architecture. Basically JORAM has adopted a snowflake architecture:

- a JORAM platform is composed of a set of JORAM servers interconnected by a message bus that offers various communication protocols.
- Each server can manage a variable number of JMS clients.

The geographical implantation of the servers as well as the distribution of the clients on the various servers are under the responsibility of the architect. This choice is a first level of configuration. A second level refers to the placement of communication objects (Queues and Topics). Note that these parameters have a deep impact on performance figures and scalability issues.

OSGi

Joram is now designed with an OSGi™ based services architecture to provide a dynamically adaptable messaging server:

- The Joram server can be launched as a set of OSGi bundles (it is always possible to launch the server as a classic Java program).
- Additional services can be started, stopped, reconfigured: Command line interface, Web console, etc.

Overview

This document is the Joram™ Reference Manual, it documents Joram 5.8.x API :

- Chapter 2 and 3 and 3 list all the classes and interfaces of the API,
- Chapter 4 describes all Joram's classes related to the JMS 1.1 unified API,
- Chapter 5 describes the manipulation of messages,
- Chapter 6 lists all the class allowing the handling of specialized destinations,
- Chapter 7 is devoted to the interception of messages,
- Chapter 8 and 9 remind notions of the old JMS 1.0 API dedicated to Point-To-Point and Publish/Subscribe interface,
- Chapter 10 is devoted to the API needed for application servers,
- Chapter 11 describes the classes devoted to the administration,
- Chapter 12 describes the configuration of Joram through XML scriptst,
- Chapter 13 describes the classes allowing the configuration of Joram through the OSGi ConfigurationAdmin,
- Chapter 14 describes the command line interface,
- Chapter 15 presents the JMX interface implemented by Joram's classes,
- Chapter 16 describes the use of Joram in Spring framework,
- Chapter 17 lists the exception's classes,
- Chapter 19 lists the environment variables to configure Joram,

- Chapter 20 lists the main error's messages,
- Finally Chapter 2112 list all deprecated classes.

2. Class Hierarchy Index

This inheritance list is sorted roughly, but not completely, alphabetically:

org.objectweb.joram.client.osgi.Activator	276
org.objectweb.joram.client.jms.admin.AdminException	206
org.objectweb.joram.client.jms.admin.AdminHelper	206
org.objectweb.joram.client.jms.ConnectionFactory	19
org.objectweb.joram.client.jms.local.LocalConnectionFactory	24
org.objectweb.joram.client.jms.tcp.TcpConnectionFactory	24
org.objectweb.joram.client.jms.local.QueueLocalConnectionFactory	187
org.objectweb.joram.client.jms.tcp.QueueTcpConnectionFactory	187
org.objectweb.joram.client.jms.local.TopicLocalConnectionFactory	196
org.objectweb.joram.client.jms.tcp.TopicTcpConnectionFactory	196
org.objectweb.joram.client.jms.local.XALocalConnectionFactory	93
org.objectweb.joram.client.jms.tcp.XATcpConnectionFactory	94
org.objectweb.joram.client.jms.local.XAQueueLocalConnectionFactory	192
org.objectweb.joram.client.jms.tcp.XAQueueTcpConnectionFactory	192
org.objectweb.joram.client.jms.local.XATopicLocalConnectionFactory	202
org.objectweb.joram.client.jms.tcp.XATopicTcpConnectionFactory	202
org.objectweb.joram.client.jms.admin.ClusterConnectionFactory	88
org.objectweb.joram.client.jms.admin.User	79
org.objectweb.joram.client.jms.Destination	57
org.objectweb.joram.client.jms.admin.ClusterDestination	89
org.objectweb.joram.client.jms.admin.ClusterQueue	91
org.objectweb.joram.client.jms.admin.ClusterTopic	91
org.objectweb.joram.client.jms.Queue	65
org.objectweb.joram.client.jms.admin.DeadMQueue	92
org.objectweb.joram.client.jms.TemporaryQueue	73
org.objectweb.joram.client.jms.Topic	74
org.objectweb.joram.client.jms.TemporaryTopic	78
org.objectweb.joram.client.jms.admin.AdminIf	Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.admin.AdminWrapper	Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.admin.JoramAdmin	291
org.objectweb.joram.client.jms.admin.AdminModule	225
org.objectweb.joram.client.jms.Connection	28
org.objectweb.joram.client.jms.QueueConnection	188
org.objectweb.joram.client.jms.XAQueueConnection	193
org.objectweb.joram.client.jms.TopicConnection	197
org.objectweb.joram.client.jms.XATopicConnection	203
org.objectweb.joram.client.jms.XAConnection	94

org.objectweb.joram.client.osgi.ConnectionFactoryMSF	286	
org.objectweb.joram.client.jms.ConnectionMetaData		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.pool.ConnectionPool		Erreur : source de la référence non trouvée
org.objectweb.joram.client.osgi.DestinationMSF	286	
org.objectweb.joram.client.jms.FactoryParameters	25	
org.objectweb.joram.client.jms.admin.JoramAdminConnectMBean		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.admin.JoramAdminConnect		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.admin.JoramAdminMBean		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.admin.JoramAdmin	291	
org.objectweb.joram.client.jms.admin.JoramSaxWrapper	142	
org.objectweb.joram.client.jms.local.LocalConnectionsMBean		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.local.LocalConnections		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.local.LocalRequestChannelMBean		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.Message	102	
org.objectweb.joram.client.jms.BytesMessage	114	
org.objectweb.joram.client.jms.MapMessage	121	
org.objectweb.joram.client.jms.ObjectMessage	134	
org.objectweb.joram.client.jms.StreamMessage	134	
org.objectweb.joram.client.jms.TextMessage	134	
org.objectweb.joram.client.jms.MessageConsumer	49	
org.objectweb.joram.client.jms.QueueReceiver	190	
org.objectweb.joram.client.jms.TopicSubscriber	201	
org.objectweb.joram.client.jms.MessageInterceptor		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.MessageProducer	52	
org.objectweb.joram.client.jms.QueueSender		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.TopicPublisher		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.MultiSessionConsumer		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.pool.PooledConnection	169	
org.objectweb.joram.client.jms.pool.PooledConnectionFactory	173	
org.objectweb.joram.client.jms.QueueBrowser		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.Session	36	
org.objectweb.joram.client.jms.QueueSession		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.TopicSession		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.admin.Subscription		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.tcp.TcpRequestChannel	180	
org.objectweb.joram.client.osgi.UserMSF	287	
org.objectweb.joram.client.jms.XAResource		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.XAResourceMngr		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.XASession		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.XAQueueSession		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.XATopicSession		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.XidImpl		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.admin.XmlSerializer		Erreur : source de la référence non trouvée
org.objectweb.joram.client.jms.admin.server.ZeroconfJoramServer		Erreur : source de la référence non trouvée

3. Class List

Here is the list of mainare the classes and interfaces with brief descriptions:

org.objectweb.joram.client.osgi.Activator	276
org.objectweb.joram.client.jms.admin.AdminException	206
org.objectweb.joram.client.jms.admin.AdminHelper	206
org.objectweb.joram.client.jms.admin.AdminItf	209
org.objectweb.joram.client.jms.admin.AdminModule	225
org.objectweb.joram.client.jms.BytesMessage	114
org.objectweb.joram.client.jms.admin.ClusterConnectionFactory	88
org.objectweb.joram.client.jms.admin.ClusterDestination	89
org.objectweb.joram.client.jms.admin.ClusterQueue	91
org.objectweb.joram.client.jms.admin.ClusterTopic	91
org.objectweb.joram.client.jms.Connection	28
org.objectweb.joram.client.jms.ConnectionFactory	19
org.objectweb.joram.client.osgi.ConnectionFactoryMSF	286
org.objectweb.joram.client.jms.ConnectionMetaData	34
org.objectweb.joram.client.jms.pool.ConnectionPool	100
org.objectweb.joram.client.jms.admin.DeadMQueue	92
org.objectweb.joram.client.jms.Destination	57
org.objectweb.joram.client.osgi.DestinationMSF	286
org.objectweb.joram.client.jms.FactoryParameters	25
org.objectweb.joram.client.jms.admin.JoramAdmin	291
org.objectweb.joram.client.jms.admin.JoramSaxWrapper	142
org.objectweb.joram.client.jms.local.LocalConnectionFactory	24
org.objectweb.joram.client.jms.MapMessage	121
org.objectweb.joram.client.jms.Message	102
org.objectweb.joram.client.jms.MessageConsumer	49
org.objectweb.joram.client.jms.MessageProducer	52
org.objectweb.joram.client.jms.ObjectMessage	134
org.objectweb.joram.client.jms.pool.PooledConnection	169
org.objectweb.joram.client.jms.pool.PooledConnectionFactory	173
org.objectweb.joram.client.jms.Queue	65
org.objectweb.joram.client.jms.QueueBrowser	56
org.objectweb.joram.client.jms.QueueConnection	188
org.objectweb.joram.client.jms.local.QueueLocalConnectionFactory	187
org.objectweb.joram.client.jms.QueueReceiver	190
org.objectweb.joram.client.jms.QueueSender	191
org.objectweb.joram.client.jms.QueueSession	189
org.objectweb.joram.client.jms.tcp.QueueTcpConnectionFactory	187
org.objectweb.joram.client.jms.Session	36
org.objectweb.joram.client.jms.StreamMessage	134
org.objectweb.joram.client.jms.tcp.TcpConnectionFactory	24
org.objectweb.joram.client.jms.TemporaryQueue	73
org.objectweb.joram.client.jms.TemporaryTopic	78
org.objectweb.joram.client.jms.TextMessage	134
org.objectweb.joram.client.jms.Topic	74
org.objectweb.joram.client.jms.TopicConnection	197
org.objectweb.joram.client.jms.local.TopicLocalConnectionFactory	196
org.objectweb.joram.client.jms.TopicPublisher	199
org.objectweb.joram.client.jms.TopicSession	198

org.objectweb.joram.client.jms.TopicSubscriber	201
org.objectweb.joram.client.jms.tcp.TopicTcpConnectionFactory	196
org.objectweb.joram.client.jms.admin.User	79
org.objectweb.joram.client.osgi.UserMSF	287
org.objectweb.joram.client.jms.XAConnection	94
org.objectweb.joram.client.jms.local.XALocalConnectionFactory	93
org.objectweb.joram.client.jms.XAQueueConnection	193
org.objectweb.joram.client.jms.local.XAQueueLocalConnectionFactory	192
org.objectweb.joram.client.jms.XAQueueSession	194
org.objectweb.joram.client.jms.tcp.XAQueueTcpConnectionFactory	192
org.objectweb.joram.client.jms.XASession	95
org.objectweb.joram.client.jms.tcp.XATcpConnectionFactory	94
org.objectweb.joram.client.jms.XATopicConnection	203
org.objectweb.joram.client.jms.local.XATopicLocalConnectionFactory	202
org.objectweb.joram.client.jms.XATopicSession	204
org.objectweb.joram.client.jms.tcp.XATopicTcpConnectionFactory	202
org.objectweb.joram.client.jms.admin.server.ZeroconfJoramServer	261

4. Unified JMS API

Class reference

This chapter describes the Joram classes and methods needed to build programs in respect with the JMS 1.1 API.

4.1. ConnectionFactory

Class org.objectweb.joram.client.jms.ConnectionFactory
 Implements javax.jms.ConnectionFactory, javax.jms.QueueConnectionFactory, javax.jms.TopicConnectionFactory,
 javax.jms.XAConnectionFactory, javax.jms.XAQueueConnectionFactory, javax.jms.XATopicConnectionFactory

Public Member Functions

```
FactoryParameters getParameters ()
javax.jms.Connection createConnection () throws JMSEException
javax.jms.Connection createConnection (String name, String password) throws JMSEException
javax.jms.QueueConnection createQueueConnection () throws JMSEException
javax.jms.QueueConnection createQueueConnection (String name, String password) throws JMSEException
javax.jms.TopicConnection createTopicConnection () throws JMSEException
javax.jms.TopicConnection createTopicConnection (String name, String password) throws JMSEException
javax.jms.XAConnection createXAConnection () throws JMSEException
javax.jms.XAConnection createXAConnection (String name, String password) throws javax.jms.JMSEException
javax.jms.XAQueueConnection createXAQueueConnection () throws JMSEException
javax.jms.XAQueueConnection createXAQueueConnection (String name, String password) throws javax.jms.JMSEException
javax.jms.XATopicConnection createXATopicConnection () throws JMSEException
javax.jms.XATopicConnection createXATopicConnection (String name, String password) throws javax.jms.JMSEException
```

Static Public Member Functions

```
static String getDefaultServerHost ()
static int getDefaultServerPort ()
static String getDefaultRootLogin ()
static String getDefaultRootPassword ()
static String getDefaultLogin ()
static String getDefaultPassword ()
```

Detailed Description

Implements all the javax.jms.ConnectionFactory interfaces.

A ConnectionFactory object encapsulates a set of configuration parameters defined by an administrator. A client needs to use it to create a connection with a Joram server.

A ConnectionFactory object is a JMS administered object containing configuration information, it is created by an administrator and later used by JMS clients. Normally the JMS clients find administered objects by looking them up in a JNDI namespace.

ConnectionFactory objects can be programmatically created using the LocalConnectionFactory.create or TcpConnectionFactory.create methods. Created objects can be then configured using FactoryParameters object.

See also:

[javax.jms.ConnectionFactory](#)
[javax.jms.QueueConnectionFactory](#)
[javax.jms.TopicConnectionFactory](#)
[javax.jms.XAConnectionFactory](#)
[javax.jms.XAQueueConnectionFactory](#)
[javax.jms.XATopicConnectionFactory](#)

Member Function Documentation

Connection ConnectionFactory.createConnection () throws JMSEException

API method, creates a connection with the default user identity. The connection is created in stopped mode.

Returns:

a newly created connection.

See also:

[javax.jms.ConnectionFactory::createConnection\(\)](#)

Exceptions:

JMSecurityException	If the default identification is incorrect.
IllegalStateException	If the server is not listening.

Connection ConnectionFactory.createConnection (String name, String password) throws JMSEException

API method, creates a connection with the specified user identity. The connection is created in stopped mode.

Parameters:

name	the caller's user name.
password	the caller's password.

Returns:

a newly created connection.

See also:

[javax.jms.ConnectionFactory::createConnection\(String, String\)](#)

Exceptions:

JMSecurityException	If the user identification is incorrect.
IllegalStateException	If the server is not listening.

QueueConnection ConnectionFactory.createQueueConnection () throws JMSEException

API method, creates a queue connection with the default user identity. The connection is created in stopped mode.

Returns:

a newly created queue connection.

See also:

[javax.jms.QueueConnectionFactory::createQueueConnection\(\)](#)

Exceptions:

JMSecurityException	If the default identification is incorrect.
IllegalStateException	If the server is not listening.

javax.jms.QueueConnection ConnectionFactory.createQueueConnection (String name, String password) throws JMSException

API method, creates a queue connection with the specified user identity. The connection is created in stopped mode.

Parameters:

name	the caller's user name.
password	the caller's password.

Returns:

a newly created queue connection.

See also:

`javax.jms.QueueConnectionFactory::createQueueConnection(String, String)`

Exceptions:

JMSecurityException	If the user identification is incorrect.
IllegalStateException	If the server is not listening.

javax.jms.TopicConnection ConnectionFactory.createTopicConnection () throws JMSException

API method, creates a topic connection with the default user identity. The connection is created in stopped mode.

Returns:

a newly created topic connection.

See also:

`javax.jms.TopicConnectionFactory::createTopicConnection()`

Exceptions:

JMSecurityException	If the default identification is incorrect.
IllegalStateException	If the server is not listening.

javax.jms.TopicConnection ConnectionFactory.createTopicConnection (String name, String password) throws JMSException

API method, creates a topic connection with the specified user identity. The connection is created in stopped mode.

Parameters:

name	the caller's user name.
password	the caller's password.

Returns:

a newly created topic connection.

See also:

`javax.jms.TopicConnectionFactory::createTopicConnection(String, String)`

Exceptions:

JMSecurityException	If the user identification is incorrect.
IllegalStateException	If the server is not listening.

javax.jms.XAConnection ConnectionFactory.createXAConnection () throws JMSException

API method, creates an XA connection with the default user identity. The connection is created in stopped mode.

Returns:

a newly created XA connection..

See also:

`javax.jms.XAConnectionFactory::createXAConnection()`

Exceptions:

JMSSecurityException	If the default identification is incorrect.
IllegalStateException	If the server is not listening.

javax.jms.XAConnection ConnectionFactory.createXAConnection (String name, String password) throws javax.jms.JMSEException

API method, creates an XA connection with the specified user identity. The connection is created in stopped mode.

Parameters:

name	the caller's user name.
password	the caller's password.

Returns:

a newly created XA connection.

See also:

javax.jms.XAConnectionFactory::createXAConnection(String, String)

Exceptions:

JMSecurityException	If the user identification is incorrect.
IllegalStateException	If the server is not listening.

javax.jms.XAQueueConnection ConnectionFactory.createXAQueueConnection () throws JMSEException

API method, creates an XA queue connection with the default user identity. The connection is created in stopped mode.

Returns:

a newly created XA queue connection..

See also:

javax.jms.XAQueueConnectionFactory::createXAQueueConnection()

Exceptions:

JMSecurityException	If the default identification is incorrect.
IllegalStateException	If the server is not listening.

javax.jms.XAQueueConnection ConnectionFactory.createXAQueueConnection (String name, String password) throws javax.jms.JMSEException

API method, creates an XA queue connection with the specified user identity. The connection is created in stopped mode.

Parameters:

name	the caller's user name.
password	the caller's password.

Returns:

a newly created XA queue connection.

See also:

javax.jms.XAQueueConnectionFactory::createXAQueueConnection(String, String)

Exceptions:

JMSecurityException	If the user identification is incorrect.
IllegalStateException	If the server is not listening.

javax.jms.XATopicConnection ConnectionFactory.createXATopicConnection () throws JMSEException

API method, creates an XA topic connection with the default user identity. The connection is created in stopped mode.

Returns:

a newly created XA topic connection..

See also:

javax.jms.XATopicConnectionFactory::createXATopicConnection()

Exceptions:

JMSSecurityException	If the default identification is incorrect.
IllegalStateException	If the server is not listening.

javax.jms.XATopicConnection ConnectionFactory.createXATopicConnection (String name, String password) throws javax.jms.JMSEException

API method, creates an XA topic connection with the specified user identity. The connection is created in stopped mode.

Parameters:

name	the caller's user name.
password	the caller's password.

Returns:

a newly created XA topic connection.

See also:

javax.jms.XATopicConnectionFactory::createXATopicConnection(String, String)

Exceptions:

JMSSecurityException	If the user identification is incorrect.
IllegalStateException	If the server is not listening.

static String ConnectionFactory.getDefaultLogin () [static]

Returns default login name for connection. Default value "anonymous" can be adjusted by setting the JoramDfltLogin property.

static String ConnectionFactory.getDefaultPassword () [static]

Returns the default login password for connection. Default value "anonymous" can be adjusted by setting the JoramDfltPassword property.

static String ConnectionFactory.getDefaultRootLogin () [static]

Returns default administrator login name for connection. Default value "root" can be adjusted by setting the JoramDfltRootLogin property.

static String ConnectionFactory.getDefaultRootPassword () [static]

Returns the default administrator login password for connection. Default value "root" can be adjusted by setting the JoramDfltRootPassword property.

static String ConnectionFactory.getDefaultServerHost () [static]

Returns default server's hostname for connection. Default value "localhost" can be adjusted by setting the JoramDfltServerHost property.

static int ConnectionFactory.getDefaultServerPort () [static]

Returns default server's port for connection. Default value 16010 can be adjusted by setting the JoramDfltServerPort property.

FactoryParameters ConnectionFactory.getParameters ()

Returns the factory's configuration parameters.

4.2. LocalConnectionFactory

Class org.objectweb.joram.client.jms.local.LocalConnectionFactory
extends org.objectweb.joram.client.jms.ConnectionFactory

Static Public Member Functions

static ConnectionFactory create ()

Detailed Description

A LocalConnectionFactory instance is a factory of local in-VM connections.
The created ConnectionFactory can be configured using the FactoryParameters.

Member Function Documentation

static ConnectionFactory LocalConnectionFactory.create () [static]

Administration method creating a ConnectionFactory instance for creating local connections.

4.3. TcpConnectionFactory

Class org.objectweb.joram.client.jms.tcp.TcpConnectionFactory
extends org.objectweb.joram.client.jms.ConnectionFactory

Static Public Member Functions

static ConnectionFactory create () throws java.net.ConnectException

static ConnectionFactory create (String host, int port)

static ConnectionFactory create (String host, int port, String reliableClass)

Detailed Description

A TcpConnectionFactory instance is a factory of TCP connections.
The created ConnectionFactory can be configured using the FactoryParameters.

Member Function Documentation

static ConnectionFactory TcpConnectionFactory.create () throws java.net.ConnectException [static]

Administration method creating a ConnectionFactory instance for creating TCP connections with the default server.

See also:

ConnectionFactory.getDefaultServerHost()

`ConnectionFactory.getDefaultServerPort()`

static ConnectionFactory TcpConnectionFactory.create (String host, int port) [static]

Administration method creating a `ConnectionFactory` instance for creating TCP connections with a given server.

Parameters:

host	Name or IP address of the server's host.
port	Server's listening port.

static ConnectionFactory TcpConnectionFactory.create (String host, int port, String reliableClass) [static]

Administration method creating a `ConnectionFactory` instance for creating TCP connections with a given server. The `reliableClass` parameter determines the underlying protocol class used to communicate with the Joram server. By default the class `org.objectweb.joram.client.jms.tcp.ReliableTcpClient` is used. In order to use SSL sockets between this parameter must be set to "`org.objectweb.joram.client.jms.tcp.ReliableSSTCPClient`".

Parameters:

host	Name or IP address of the server's host.
port	Server's listening port.
reliableClass	Reliable class name.

4.4. FactoryParameters

Class `org.objectweb.joram.client.jms.FactoryParameters`

Public Member Functions

```
String getHost ()
int getPort ()
String getUrl ()
FactoryParameters (String host, int port)
FactoryParameters (String url)
FactoryParameters ()
```

Public Attributes

```
boolean TcpNoDelay = true
int SoLinger = -1
int SoTimeout = 0
int connectingTimer = 30
int txPendingTimer = 0
int cnxPendingTimer = 0
String socketFactory = SocketFactory.DefaultFactory
boolean implicitAck
boolean asyncSend = false
int queueMessageReadMax = 1
int topicAckBufferMax = 10
boolean multiThreadSync = false
int multiThreadSyncDelay = 1
int multiThreadSyncThreshold = 10
int topicPassivationThreshold = Integer.MAX_VALUE
int topicActivationThreshold = 0
String outLocalAddress = null
int outLocalPort = 0
int compressedMinSize = 0
int compressionLevel = Deflater.BEST_SPEED
String messageIdPrefix = null
```

Detailed Description

A FactoryParameters instance holds a <XA>ConnectionFactory configuration parameters, it allows to configure the related ConnectionFactory.

Member Function Documentation

String FactoryParameters.getHost ()

Returns the name of host hosting the server to create connections with.

Returns:

The name of host hosting the server.

int FactoryParameters.getPort ()

Returns the port to be used for accessing the server.

Returns:

The port to be used for accessing the server.

String FactoryParameters.getUrl ()

Returns the url to be used for accessing the server.

Returns:

The url to be used for accessing the server.

Member Data Documentation

boolean FactoryParameters.asyncSend = false

Determines whether the persistent produced messages are asynchronously sent (without acknowledge) or not. Messages sent asynchronously may be lost if a failure occurs before the message is persisted on the server. Non persistent messages are always sent without acknowledgment. Default is false, persistent messages are sent with acknowledge.

int FactoryParameters.cnxPendingTimer = 0

Period in milliseconds between two ping requests sent by the client connection to the server; if the server does not receive any ping request during more than $3 * \text{cnxPendingTimer}$, the connection is considered as dead and processed as required.

int FactoryParameters.connectingTimer = 30

Duration in seconds during which connecting is attempted (connecting might take time if the server is temporarily not reachable). If the value is set to 0, the client tries to connect once and abort if the connection fails. The client will be blocked until the connection is established or a TCP error occurs.

Default value is 30, the client try to connect to the server during 30 seconds.

boolean FactoryParameters.implicitAck

Determines whether the messages consumed are implicitly acknowledged or not. When true messages are immediately removed from queue when delivered.

List **FactoryParameters.inInterceptors**

List of Message interceptors while receiving a message. interceptors are called when Session::receive() is called. execution follows the order of the elements within the list. property is facultative. If set, then the MessageInterceptor.handle callback method of the IN interceptors are called.

boolean FactoryParameters.multiThreadSync = false

Determines whether client threads which are using the same connection are synchronized in order to group together the requests they send.

int FactoryParameters.multiThreadSyncDelay = 1

The maximum time the threads hang if 'multiThreadSync' is true.
Either they wake up (wait time out) or they are notified (by the first waken up thread).
Default value is 1ms.

int FactoryParameters.multiThreadSyncThreshold = 10

The maximum numbers of threads that hang if 'multiThreadSync' is true.
Default value is 10 waiting threads.

List **FactoryParameters.outInterceptors**

List of Message interceptors while sending a message. interceptors are called when Session::send() is called. execution follows the order of the elements within the list. property is facultative. If set, then the MessageInterceptor.handle callback method of the OUT interceptors are called.

String FactoryParameters.outLocalAddress = null

This is the local IP address on which the TCP connection is activated.
The value can either be a machine name, such as "java.sun.com", or a textual representation of its IP address.

int FactoryParameters.outLocalPort = 0

This is the local IP address port on which the TCP connection is activated

int FactoryParameters.queueMessageReadMax = 1

The maximum number of messages that can be read at once from a queue.
Default is 1.

String FactoryParameters.socketFactory = SocketFactory.DefaultFactory

Allows to define a specific factory for socket in order to by-pass compatibility problem between JDK version. Currently there is two factories, The default factory one for JDK since 1.4, and SocketFactory13 for JDK prior to 1.4.

int FactoryParameters.SoLinger = -1

Enable SO_LINGER with the specified linger time in seconds, if the value is less than 0 then it disables SO_LINGER. Default value is -1.

int FactoryParameters.SoTimeout = 0

Enable/disable SO_TIMEOUT with the specified timeout in milliseconds. The timeout must be > 0. A timeout of zero is interpreted as an infinite timeout. Default value is 0.

boolean FactoryParameters.TcpNoDelay = true

Enable/disable TCP_NODELAY (disable/enable Nagle's algorithm), default value is true.

int FactoryParameters.topicAckBufferMax = 10

The maximum number of acknowledgements that can be buffered in Session.DUPS_OK_ACKNOWLEDGE mode when listening to a topic.
Default value is 10.

int FactoryParameters.topicActivationThreshold = 0

This threshold is the minimum messages number below which the subscription is activated.
Default value is 0.

int FactoryParameters.topicPassivationThreshold = Integer.MAX_VALUE

This threshold is the maximum messages number over which the subscription is passivated.
Default is Integer.MAX_VALUE.

int FactoryParameters.txPendingTimer = 0

Duration in seconds during which a JMS transacted (non XA) session might be pending; above that duration the session is rolled back and closed; the 0 value means "no timer".
The default value is 0 (no timer).

int compressedMinSize = 0

This attribute defines the minimum size beyond which the message body is compressed.
The default value is 0 (no compression).

int compressionLevel = Deflater.BEST_SPEED;

This attribute defines the compression level (0-9) used when the message body is compressed.
The default value is 1 (Deflater.BEST_SPEED).

String messageIdPrefix = null;

This attribute allows to customize the JMS MessageID adding the specified string.
Classically a JMS MessageID has the following structure "**ID:<proxyid>c<key>m<order>**" where <proxyid> identifies the user, <key> is the identifier of the connection and <order> is the serial number of the message in this connection. If the property is set to <prefix> then the JMS MessageID has the following structure "**ID:<prefix>_<proxyid>c<key>m<order>**".

Be careful, the JMS MessageID is part of the JMS message header, therefore its size has an impact on the volume of data transferred and stored on the server.

This value is overloaded by the environment property "org.objectweb.joram.client.jms.messageid.prefix" if it defined.
By default undefined.

4.5. Connection

Class org.objectweb.joram.client.jms.Connection
Implements javax.jms.Connection

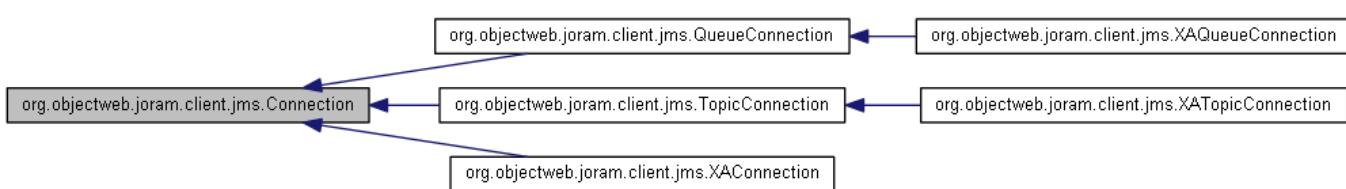


Figure 4.1 - Inheritance diagram for Connection class

Public Member Functions

`final long getTxPendingTimer ()`

```

final boolean getImplicitAck ()
final boolean getAsyncSend ()
final int getCompressedMinSize()
final int getCompressionLevel()
final int getQueueMessageReadMax ()
final int getTopicAckBufferMax ()
final int getTopicPassivationThreshold ()
final int getTopicActivationThreshold ()
final String getOutLocalAddress ()
final int getOutLocalPort ()
synchronized javax.jms.ConnectionConsumer createConnectionConsumer (javax.jms.Destination dest, String
selector, javax.jms.ServerSessionPool sessionPool, int maxMessages) throws JMSEException
javax.jms.ConnectionConsumer createDurableConnectionConsumer (javax.jms.Topic topic, String subName, String
selector, javax.jms.ServerSessionPool sessPool, int maxMessages) throws JMSEException
synchronized javax.jms.Session createSession (boolean transacted, int acknowledgeMode) throws JMSEException
synchronized void setExceptionListener (javax.jms.ExceptionListener listener) throws JMSEException
javax.jms.ExceptionListener getExceptionListener () throws JMSEException
void setClientID (String clientID) throws JMSEException
String getClientID () throws JMSEException
javax.jms.ConnectionMetaData getMetaData () throws JMSEException
synchronized void start () throws JMSEException
void stop () throws JMSEException
void close () throws JMSEException

```

Detailed Description

Implements the javax.jms.Connection interface.

A Connection object allows the client's active connection to the Joram server. Connections support concurrent use, it serves several purposes:

- It encapsulates the real connection with the Joram server (for example an open TCP/IP socket between the client and the server).
- It needs the client authentication.
- It specifies a unique client identifier.
- It supports the ExceptionListener object.

A Joram client typically creates a connection, one or more sessions, and a number of message producers and consumers.

When a connection is created, it is in stopped mode that means that no messages are being delivered. In order to minimize any client confusion that may result from asynchronous message delivery during setup, it is typical to leave the connection in stopped mode until setup is complete. A message producer can send messages while a connection is stopped.

Member Function Documentation

void org.objectweb.joram.client.jms.Connection.close () throws JMSEException

API method for closing the connection; even if the connection appears to be broken, closes the sessions.

In order to free significant resources allocated on behalf of a connection, clients should close connections when they are not needed. Closing a connection automatically close all related sessions, producers, and consumers and causes all temporary destinations to be deleted.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

See also:

javax.jms.Connection::close()

synchronized javax.jms.ConnectionConsumer Connection.createConnectionConsumer (javax.jms.Destination dest, String selector, javax.jms.ServerSessionPool sessionPool, int maxMessages) throws JMSEException

API method. Creates a connection consumer for this connection, this is an expert facility needed for applications servers.

Parameters:

dest	the destination to access.
selector	only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for this message consumer.
sessionPool	the server session pool to associate with this connection consumer
maxMessages	the maximum number of messages that can be assigned to a server session at one time.

Returns:

The connection consumer.

Exceptions:

IllegalStateException	If the connection is closed.
InvalidSelectorException	If the selector syntax is wrong.
InvalidDestinationException	If the target destination does not exist.
JMSEException	If the method fails for any other reason.

javax.jms.ConnectionConsumer Connection.createDurableConnectionConsumer (javax.jms.Topic topic, String subName, String selector, javax.jms.ServerSessionPool sessPool, int maxMessages) throws JMSEException

API method. Creates a durable connection consumer for this connection, this is an expert facility needed for applications servers.

Parameters:

topic	the topic to access.
subName	durable subscription name.
selector	only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for this message consumer.
sessionPool	the server session pool to associate with this connection consumer
maxMessages	the maximum number of messages that can be assigned to a server session at one time.

Returns:

The durable connection consumer.

Exceptions:

IllegalStateException	If the connection is closed.
InvalidSelectorException	If the selector syntax is wrong.
InvalidDestinationException	If the target topic does not exist.
JMSEException	If the method fails for any other reason.

synchronized javax.jms.Session Connection.createSession (boolean transacted, int acknowledgeMode) throws JMSEException

API method. Creates a Session object.

Parameters:

transacted	indicates whether the session is transacted.
acknowledgeMode	indicates whether the consumer or the client will acknowledge any messages it receives; ignored if the session is transacted. Legal values defined by the JMS specification are: <ul style="list-style-type: none">• Session.AUTO_ACKNOWLEDGE,

	<ul style="list-style-type: none"> • Session.CLIENT_ACKNOWLEDGE, • and Session.DUPS_OK_ACKNOWLEDGE. <p>Additionnaly Joram defines INDIVIDUAL_ACKNOWLEDGE: With this acknowledgment mode, the client acknowledges a consumed message by calling the message's acknowledge method. Contrary to CLIENT_ACKNOWLEDGE mode this mode allows to acknowledge only the specified message.</p>
--	--

Returns:

A newly created session.

Exceptions:

IllegalStateException	If the connection is closed.
JMSEException	In case of an invalid acknowledge mode.

boolean Connection.equals (Object obj)

Returns true if the parameter is a Connection instance sharing the same proxy identifier and connection key.

final boolean Connection.getAsyncSend ()

Indicates whether the persistent produced messages are asynchronously sent (without acknowledge) or not. Messages sent asynchronously may be lost if a failure occurs before the message is persisted on the server.

Non persistent messages are always sent without acknowledgment.

This attribute is inherited from FactoryParameters, by default false, persistent messages are sent with acknowledge.

Returns:

true if messages produced are asynchronously sent.

See also:

FactoryParameters::asyncSend

Session::isAsyncSend()

String Connection.getClientID () throws JMSEException

API method. Gets the client identifier for this connection. This value is specific to the Joram, it is automatically assigned by the server.

Returns:

the unique client identifier.

Exceptions:

IllegalStateException	If the connection is closed.
-----------------------	------------------------------

final int Connection.getCompressedMinSize()

Returns the minimum size beyond which the message body is compressed.

The default value is 0 (no compression).

Returns:

the minimum size beyond which the message body is compressed.

See also:

FactoryParameters::compressedMinSize

final int Connection.getCompressionLevel()

Get the compression level for this Connection, this attribute is inherited from FactoryParameters. The default value is 1 (Deflater.BEST_SPEED).

Returns:

The compression level.

See also:

FactoryParameters::compressionLevel

[javax.jms.ExceptionListener Connection.getExceptionListener \(\) throws JMSEException](#)

API method. Gets the ExceptionListener object for this connection if it is configured.

Returns:

the ExceptionListener for this connection, or null, if no ExceptionListener is associated with this connection.

Exceptions:

IllegalStateException	If the connection is closed.
-----------------------	------------------------------

[final boolean Connection.getImplicitAck \(\)](#)

Indicates whether the messages consumed are implicitly acknowledged or not. If true messages are immediately removed from queue when delivered and there is none acknowledge message from client to server.

This attribute is inherited from FactoryParameters, by default false.

Returns:

true if messages produced are implicitly acknowledged.

See also:

FactoryParameters::implicitAck
Session::isImplicitAck()

[javax.jms.ConnectionMetaData Connection.getMetaData \(\) throws JMSEException](#)

API method. Gets the metadata for this connection.

Returns:

the connection metadata.

Exceptions:

IllegalStateException	If the connection is closed.
-----------------------	------------------------------

See also:

ConnectionMetaData

[final String Connection.getOutLocalAddress \(\)](#)

Returns the local IP address on which the TCP connection is activated.

This attribute is inherited from FactoryParameters.

Returns:

the local IP address on which the TCP connection is activated.

See also:

FactoryParameters::outLocalAddress

[final int Connection.getOutLocalPort \(\)](#)

Returns the local IP address port on which the TCP connection is activated

This attribute is inherited from FactoryParameters.

Returns:

the local IP address port on which the TCP connection is activated.

See also:

FactoryParameters::outLocalPort

final int Connection.getQueueMessageReadMax ()

Get the maximum number of messages that can be read at once from a queue for this Connection.
This attribute is inherited from FactoryParameters, default value is 1.

Returns:

The maximum number of messages that can be read at once from a queue.

See also:

FactoryParameters::queueMessageReadMax
Session::getQueueMessageReadMax()

final int Connection.getTopicAckBufferMax ()

Get the maximum number of acknowledgements that can be buffered when using Session.DUPS_OK_ACKNOWLEDGE mode for this Connection.
This attribute is inherited from FactoryParameters, default value is 0.

Returns:

The Maximum number of acknowledgements that can be buffered when using Session.DUPS_OK_ACKNOWLEDGE mode.

See also:

FactoryParameters::topicAckBufferMax
Session::getTopicAckBufferMax()

final int Connection.getTopicActivationThreshold ()

Get the threshold of activation for this Connection.
This threshold is the minimum messages number below which the subscription is activated.
This attribute is inherited from FactoryParameters, default value is 0.

Returns:

The minimum messages number below which the subscription is activated.

See also:

FactoryParameters::topicActivationThreshold
Session::getTopicActivationThreshold()

final int Connection.getTopicPassivationThreshold ()

Get the threshold of passivation for this Connection.
This threshold is the maximum messages number over which the subscription is passivated.
This attribute is inherited from FactoryParameters, default value is Integer.MAX_VALUE.

Returns:

The maximum messages number over which the subscription is passivated.

See also:

FactoryParameters::topicPassivationThreshold
Session::getTopicPassivationThreshold()

final long Connection.getTxPendingTimer ()

Returns the duration in seconds during which a JMS transacted (non XA) session might be pending; above that duration the session is rolled back and closed; the 0 value means "no timer".

Returns:

the duration in seconds during which a JMS transacted (non XA) session might be pending.

See also:

[FactoryParameters::txPendingTimer](#)

void Connection.setClientID (String clientID) throws JMSException

API method. Sets the client identifier for this connection. Joram automatically allocates a client identifier when the connection is created, so this method always throws an `IllegalStateException`.

Parameters:

clientID	the unique client identifier.
----------	-------------------------------

Exceptions:

<code>IllegalStateException</code>	Systematically thrown.
------------------------------------	------------------------

synchronized void Connection.setExceptionListener (javax.jms.ExceptionListener listener) throws JMSException

API method. Sets an exception listener for this connection. When Joram detects a serious problem with a connection, it informs the connection's `ExceptionListener`, if one has been registered. It does this by calling the listener's `onException` method, passing it a `JMSException` object describing the problem.

The exception listener allows a client to be notified of a problem asynchronously. Some connections only consume messages, so they would have no other way to learn their connection has failed. A connection serializes execution of its `ExceptionListener`.

Parameters:

listener	the exception listener.
----------	-------------------------

Exceptions:

<code>IllegalStateException</code>	If the connection is closed.
------------------------------------	------------------------------

synchronized void Connection.start () throws JMSException

API method for starting the connection.

Starts (or restarts) the connection's delivery of incoming messages. A call to start on a connection that has already been started is ignored.

Exceptions:

<code>IllegalStateException</code>	If the connection is closed or broken.
------------------------------------	--

void Connection.stop () throws JMSException

API method for stopping the connection; even if the connection appears to be broken, stops the sessions.

Temporarily stops the connection's delivery of incoming messages. Delivery can be restarted using the connection's `start` method. When the connection is stopped, delivery to all the connection's message consumers is inhibited.

Stopping a connection has no effect on its ability to send messages. A call to stop on a connection that has already been stopped is ignored. This call blocks until receives and/or message listeners in progress have completed, it must not return until delivery of messages has paused.

Exceptions:

<code>IllegalStateException</code>	If the connection is closed or broken.
------------------------------------	--

4.6. ConnectionMetaData

Class org.objectweb.joram.client.jms.ConnectionMetaData
Implements javax.jms.ConnectionMetaData

Public Member Functions

```
int getJMSMajorVersion () throws JMSException
int getJMSMinorVersion () throws JMSException
```

```
String getJMSVersion () throws JMSEException
String getJMSProviderName () throws JMSEException
Enumeration getJMSXPropertyNames () throws JMSEException
int getProviderMajorVersion () throws JMSEException
int getProviderMinorVersion () throws JMSEException
String getProviderVersion () throws JMSEException
```

Static Public Attributes

```
static final int jmsMajorVersion = 1
static final int jmsMinorVersion = 1
static final String jmsVersion = "1.1"
static final String providerName = "Joram"
static final int providerMajorVersion
static final int providerMinorVersion
static final String providerVersion
```

Detailed Description

Implements the javax.jms.ConnectionMetaData interface.

Member Function Documentation

int ConnectionMetaData.getJMSMajorVersion () throws JMSEException

API method: Gets the JMS major version number.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

int ConnectionMetaData.getJMSMinorVersion () throws JMSEException

API method: Gets the JMS minor version number

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

String ConnectionMetaData.getJMSProviderName () throws JMSEException

API method: Gets the JMS provider name: Joram.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

String ConnectionMetaData.getJMSVersion () throws JMSEException

API method: Gets the JMS API version, currently 1.1.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

Enumeration ConnectionMetaData.getJMSXPropertyNames () throws JMSEException

API method: Gets an enumeration of the JMSX property names, currently JMSXDeliveryCount, JMSXGroupID and JMSXGroupSeq..

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

int ConnectionMetaData.getProviderMajorVersion () throws JMSEException

API method: Gets the Joram's major version number.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

int ConnectionMetaData.getProviderMinorVersion () throws JMSEException

API method: Gets the Joram's minor version number.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

String ConnectionMetaData.getProviderVersion () throws JMSEException

API method: Gets the Joram's implementation version.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

Member Data Documentation

final int ConnectionMetaData.jmsMajorVersion = 1 [static]

JMS major version number.

final int ConnectionMetaData.jmsMinorVersion = 1 [static]

JMS minor version number .

final String ConnectionMetaData.jmsVersion = "1.1" [static]

JMS API version, currently "1.1".

final int ConnectionMetaData.providerMajorVersion [static]

Joram's major version number.

final int ConnectionMetaData.providerMinorVersion [static]

Joram's minor version number.

final String ConnectionMetaData.providerName = "Joram" [static]

JMS provider name: Joram.

final String ConnectionMetaData.providerVersion [static]

Joram's implementation version.

4.7. Session

Class org.objectweb.joram.client.jms.Session

implements javax.jms.session

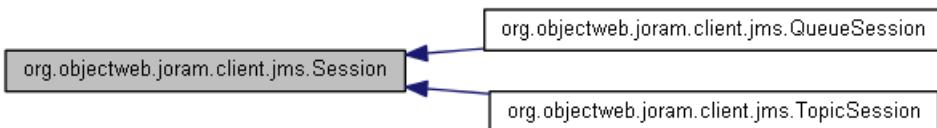


Figure 4.2 - Inheritance diagram for Session class

Public Member Functions

```

boolean isImplicitAck ()
void setImplicitAck (boolean implicitAck)
boolean isAsyncSend ()
void setAsyncSend (boolean asyncSend)
final int getQueueMessageReadMax ()
void setQueueMessageReadMax (int queueMessageReadMax)
final int getTopicAckBufferMax ()
void setTopicAckBufferMax (int topicAckBufferMax)
final int getTopicPassivationThreshold ()
void setTopicPassivationThreshold (int topicPassivationThreshold)
final int getTopicActivationThreshold ()
void setTopicActivationThreshold (int topicActivationThreshold)
boolean isAsyncSub ()
void setAsyncSub (boolean asyncSub)
final int getCompressedMinSize()
final void setCompressedMinSize(int compressedMinSize)
final int getCompressionLevel()
final void setCompressionLevel(int compressionLevel)
final int getAcknowledgeMode () throws JMSEException
synchronized final boolean getTransacted () throws JMSEException
synchronized void setMessageListener (javax.jms.MessageListener messageListener) throws JMSEException
synchronized javax.jms.MessageListener getMessageListener () throws JMSEException
synchronized javax.jms.Message createMessage () throws JMSEException
synchronized javax.jms.TextMessage createTextMessage () throws JMSEException
synchronized javax.jms.TextMessage createTextMessage (String text) throws JMSEException
synchronized javax.jms.BytesMessage createBytesMessage () throws JMSEException
synchronized javax.jms.MapMessage createMapMessage () throws JMSEException
synchronized javax.jms.ObjectMessage createObjectMessage () throws JMSEException
synchronized javax.jms.ObjectMessage createObjectMessage (java.io.Serializable obj) throws JMSEException
synchronized javax.jms.StreamMessage createStreamMessage () throws JMSEException
synchronized javax.jms.QueueBrowser createBrowser (javax.jms.Queue queue, String selector) throws JMSEException
synchronized javax.jms.QueueBrowser createBrowser (javax.jms.Queue queue) throws JMSEException
synchronized javax.jms.MessageProducer createProducer (javax.jms.Destination dest) throws JMSEException
synchronized javax.jms.MessageConsumer createConsumer (javax.jms.Destination dest, String selector, boolean noLocal) throws JMSEException
synchronized javax.jms.MessageConsumer createConsumer (javax.jms.Destination dest, String selector) throws JMSEException
synchronized javax.jms.MessageConsumer createConsumer (javax.jms.Destination dest) throws JMSEException
synchronized javax.jms.TopicSubscriber createDurableSubscriber (javax.jms.Topic topic, String name, String selector, boolean noLocal) throws JMSEException
synchronized javax.jms.TopicSubscriber createDurableSubscriber (javax.jms.Topic topic, String name) throws JMSEException
synchronized javax.jms.Queue createQueue (String name) throws JMSEException
synchronized javax.jms.Topic createTopic (String name) throws JMSEException
synchronized javax.jms.TemporaryQueue createTemporaryQueue () throws JMSEException
synchronized javax.jms.TemporaryTopic createTemporaryTopic () throws JMSEException
synchronized void run ()
synchronized void commit () throws JMSEException
synchronized void rollback () throws JMSEException
  
```

```
synchronized void recover () throws JMSException
synchronized void unsubscribe (String name) throws JMSException
void close () throws JMSException
```

Detailed Description

Implements the javax.jms.Session interface.

A Session object is a single-threaded context for producing and consuming messages. A session serves several purposes:

- It is a factory for message producers and consumers.
- It is a factory for Joram specific message.
- It defines a serial order for the messages it consumes and the messages it produces.
- It retains messages it consumes until they have been acknowledged.
- It serializes execution of message listeners registered with its message consumers.
- It is a factory for TemporaryTopics and TemporaryQueues.
- It supports a single series of transactions that combine work spanning its producers and consumers into atomic units.

A session can create and service multiple message producers and consumers.

The Session class defines the different acknowledge modes:

- AUTO_ACKNOWLEDGE – With this acknowledgment mode, the session automatically acknowledges a client's receipt of a message either when the session has successfully returned from a call to receive or when the message listener the session has called to process the message successfully returns.
- CLIENT_ACKNOWLEDGE – With this acknowledgment mode, the client acknowledges a consumed message by calling the message's acknowledge method.
- DUPS_OK_ACKNOWLEDGE – This acknowledgment mode instructs the session to lazily acknowledge the delivery of messages.
- SESSION_TRANSACTED – This value is returned from the method getAcknowledgeMode if the session is transacted.

Member Function Documentation

void Session.close () throws JMSException

API method. Closes the session.

In order to free significant resources allocated on behalf of a session, clients should close sessions when they are not needed. Closing a session automatically close all related producers, and consumers and causes all temporary destinations to be deleted.

This call will block until a receive call or message listener in progress has completed. A blocked message consumer receive call returns null when this session is closed. Closing a transacted session must roll back the transaction in progress.

This method is the only Session method that can be called concurrently.

Invoking any other Session method on a closed session must throw a JMSException.IllegalStateException. Closing a closed session must not throw an exception.

Exceptions:

JMSException	if the JMS provider fails to close the session due to some internal error.
--------------	--

synchronized void Session.commit () throws JMSException

API method. Commits all messages done in this transaction and releases any locks currently held.

Exceptions:

IllegalStateException	If the session is closed, or not transacted, or if the connection is broken.
-----------------------	--

synchronized javax.jms.QueueBrowser Session.createBrowser (javax.jms.Queue queue, String selector) throws JMSException

API method. Creates a QueueBrowser object to peek at the messages on the specified queue using a message selector.

Parameters:

queue	the queue to browse
selector	the expression allowing to filter messages

Returns:

a QueueBrowser object.

Exceptions:

IllegalStateException	if the session is closed.
InvalidDestinationException	if an invalid destination is specified.
InvalidSelectorException	if the message selector is invalid.

synchronized javax.jms.QueueBrowser Session.createBrowser (javax.jms.Queue queue) throws JMSException

API method. Creates a QueueBrowser object to peek at the messages on the specified queue.

Parameters:

queue	the queue to browse
-------	---------------------

Returns:

a QueueBrowser object.

Exceptions:

IllegalStateException	if the session is closed.
InvalidDestinationException	if an invalid destination is specified.

synchronized javax.jms.BytesMessage Session.createBytesMessage () throws JMSException

API method. Creates a BytesMessage object, a BytesMessage object could be used to send a message containing a stream of uninterpreted bytes.

Returns:

a newly created BytesMessage object.

Exceptions:

IllegalStateException	If the session is closed.
-----------------------	---------------------------

See also:

BytesMessage

synchronized javax.jms.MessageConsumer Session.createConsumer (javax.jms.Destination dest, String selector, boolean noLocal) throws JMSException

API method. Creates a MessageConsumer for the specified destination using a message selector. A client uses a MessageConsumer object to receive messages that have been sent to a destination.

In some cases, a connection may both publish and subscribe to a topic. The consumer NoLocal attribute allows a consumer to inhibit the delivery of messages published by its own connection. The default value for this attribute is False. The noLocal value is only supported by destinations that are topics.

Parameters:

dest	the Destination to access.
selector	The selector allowing to filter messages.
noLocal	if true, and the destination is a topic, inhibits the delivery of messages published by its own connection.

Returns:

the created MessageConsumer object.

Exceptions:

InvalidDestinationException	if an invalid destination is specified.
IllegalStateException	If the session is closed or if the connection is broken.
JMSEException	If the creation fails for any other reason.

synchronized javax.jms.MessageConsumer Session.createConsumer (javax.jms.Destination dest, String selector) throws JMSEException

API method. Creates a MessageConsumer for the specified destination using a message selector. A client uses a MessageConsumer object to receive messages that have been sent to a destination.

Parameters:

dest	the Destination to access.
selector	The selector allowing to filter messages.

Returns:

the created MessageConsumer object.

Exceptions:

InvalidDestinationException	if an invalid destination is specified.
IllegalStateException	If the session is closed or if the connection is broken.
JMSEException	If the creation fails for any other reason.

synchronized javax.jms.MessageConsumer Session.createConsumer (javax.jms.Destination dest) throws JMSEException

API method. Creates a MessageConsumer for the specified destination. A client uses a MessageConsumer object to receive messages that have been sent to a destination.

Parameters:

dest	the Destination to access.
------	----------------------------

Returns:

the created MessageConsumer object.

Exceptions:

InvalidDestinationException	if an invalid destination is specified.
IllegalStateException	If the session is closed or if the connection is broken.
JMSEException	If the creation fails for any other reason.

synchronized javax.jms.TopicSubscriber Session.createDurableSubscriber (javax.jms.Topic topic, String name, String selector, boolean noLocal) throws JMSEException

API method. Creates a durable subscriber to the specified topic, using a message selector and specifying whether messages published by its own connection should be delivered to it.

If a client needs to receive all the messages published on a topic, including the ones published while the subscriber is inactive, it needs to use a durable TopicSubscriber. Joram retains a record of durable subscriptions and insures that all messages from the topic's publishers are retained until they are acknowledged by this durable subscriber or they have expired.

A client can change an existing durable subscription by creating a durable TopicSubscriber with the same name and a new topic and/or message selector. Changing a durable subscriber is equivalent to unsubscribing (deleting) the old one and creating a new one.

Parameters:

topic	the non-temporary Topic to subscribe to.
name	the name used to identify this subscription.
selector	The selector allowing to filter messages. A value of null or an empty string indicates that there is no message selector for the message consumer.
noLocal	if true, inhibits the delivery of messages published by its own connection.

Returns:

the created TopicSubscriber object.

Exceptions:

InvalidDestinationException	if an invalid destination is specified.
IllegalStateException	If the session is closed or if the connection is broken.
JMSEException	If the creation fails for any other reason.

synchronized javax.jms.TopicSubscriber Session.createDurableSubscriber (javax.jms.Topic topic, String name) throws JMSEException

API method. Creates a durable subscriber to the specified topic.

If a client needs to receive all the messages published on a topic, including the ones published while the subscriber is inactive, it needs to use a durable TopicSubscriber. Joram retains a record of durable subscriptions and insures that all messages from the topic's publishers are retained until they are acknowledged by this durable subscriber or they have expired.

A client can change an existing durable subscription by creating a durable TopicSubscriber with the same name and a new topic and/or message selector. Changing a durable subscriber is equivalent to unsubscribing (deleting) the old one and creating a new one.

Parameters:

topic	the non-temporary Topic to subscribe to.
name	the name used to identify this subscription.

Returns:

the created TopicSubscriber object.

Exceptions:

InvalidDestinationException	if an invalid destination is specified.
IllegalStateException	If the session is closed or if the connection is broken.
JMSEException	If the creation fails for any other reason.

synchronized javax.jms.MapMessage Session.createMapMessage () throws JMSEException

API method. Creates a MapMessage object, a MapMessage object is used to send a set of name-value pairs, where names are String objects and values are primitive values.

Returns:

a newly created MapMessage object.

Exceptions:

IllegalStateException	If the session is closed.
-----------------------	---------------------------

See also:

MapMessage

synchronized javax.jms.ObjectMessage Session.createObjectMessage () throws JMSEException

API method. Creates an ObjectMessage object, an ObjectMessage object is used to send a message that contains a serializable Java object.

Returns:

a newly created ObjectMessage object.

Exceptions:

IllegalStateException	If the session is closed.
-----------------------	---------------------------

See also:

ObjectMessage

synchronized javax.jms.ObjectMessage Session.createObjectMessage (java.io.Serializable obj) throws JMSException

API method. Creates an ObjectMessage object, an ObjectMessage object is used to send a message that contains a serializable Java object.

Parameters:

object	the object to use to initialize this message.
--------	---

Returns:

a newly created ObjectMessage object.

Exceptions:

IllegalStateException	If the session is closed.
-----------------------	---------------------------

See also:

ObjectMessage

synchronized javax.jms.MessageProducer Session.createProducer (javax.jms.Destination dest) throws JMSException

API method. Creates a MessageProducer to send messages to the specified destination. A client uses a MessageProducer object to send messages to a destination.

Parameters:

dest	the Destination to send to, or null if this is a producer which does not have a specified destination.
------	--

Returns:

the created MessageProducer object.

Exceptions:

InvalidDestinationException	if an invalid destination is specified.
IllegalStateException	If the session is closed or if the connection is broken.
JMSException	If the creation fails for any other reason.

synchronized javax.jms.Queue Session.createQueue (String name) throws JMSException

API method. This method allows to create or retrieve a Queue with the given name on the local server. First a destination with the specified name is searched on the server, if it does not exist it is created. In any case a queue identity with its Joram specific address is returned.

If the given name is a provider-specific name (#x.y.z unique identifier) a queue identity is returned with the specified identifier.

Clients that depend on this ability are not portable. Normally the physical creation of destination is an administrative task and is not to be initiated by the JMS API.

Parameters:

name	the name of this queue.
------	-------------------------

Returns:

a queue with the given name.

Exceptions:

IllegalStateException	If the session is closed.
JMSException	If the topic creation failed.

See also:

Queue

synchronized javax.jms.StreamMessage Session.createStreamMessage () throws JMSException

API method. Creates a StreamMessage object, a StreamMessage object is used to send a self-defining stream of primitive values.

Returns:

a newly created StreamMessage object.

Exceptions:

<code>IllegalStateException</code>	If the session is closed.
------------------------------------	---------------------------

See also:

`StreamMessage`

synchronized `javax.jms.TemporaryQueue` `Session.createTemporaryQueue ()` throws `JMSEException`

API method. Creates a TemporaryQueue object. Its lifetime will be that of the Connection unless it is deleted earlier.

Returns:

a temporary queue identity.

Exceptions:

<code>IllegalStateException</code>	If the session is closed or if the connection is broken.
<code>JMSEException</code>	If the request fails for any other reason.

See also:

`TemporaryQueue`

synchronized `javax.jms.TemporaryTopic` `Session.createTemporaryTopic ()` throws `JMSEException`

API method. Creates a TemporaryTopic object. Its lifetime will be that of the Connection unless it is deleted earlier.

Returns:

a temporary topic identity.

Exceptions:

<code>IllegalStateException</code>	If the session is closed or if the connection is broken.
<code>JMSEException</code>	If the request fails for any other reason.

See also:

`TemporaryTopic`

synchronized `javax.jms.TextMessage` `Session.createTextMessage ()` throws `JMSEException`

API method. Creates a TextMessage object, a TextMessage object is used to send a message containing a String object.

Returns:

a newly created TextMessage object.

Exceptions:

<code>IllegalStateException</code>	If the session is closed.
------------------------------------	---------------------------

See also:

`TextMessage`

synchronized `javax.jms.TextMessage` `Session.createTextMessage (String text)` throws `JMSEException`

API method. Creates a TextMessage object, a TextMessage object is used to send a message containing a String object.

Parameters:

<code>text</code>	the string to use to initialize this message.
-------------------	---

Returns:

a newly created TextMessage object.

Exceptions:

IllegalStateException	If the session is closed.
-----------------------	---------------------------

See also:

TextMessage

synchronized javax.jms.Topic Session.createTopic (String name) throws JMSEException

API method. This method allows to create or retrieve a Topic with the given name on the local server. First a destination with the specified name is searched on the server, if it does not exist it is created. In any case a topic identity with its provider-specific address is returned.

If the given name is a Joram specific name (#x.y.z unique identifier) a topic identity is returned with the specified identifier.

Clients that depend on this ability are not portable. Normally the physical creation of destination is an administrative task and is not to be initiated by the JMS API.

Parameters:

name	the name of this topic.
------	-------------------------

Returns:

a topic with the given name.

Exceptions:

IllegalStateException	If the session is closed.
JMSEException	If the topic creation failed.

See also:

Topic

final int Session.getAcknowledgeMode () throws JMSEException

API method. Returns the acknowledgement mode of the session. The acknowledgement mode is set at the time that the session is created. If the session is transacted, the acknowledgement mode is ignored.

Returns:

If the session is not transacted, returns the current acknowledgement mode for the session. If the session is transacted, returns Session.SESSION_TRANSACTED.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

synchronized javax.jms.MessageListener Session.getMessageListener () throws JMSEException

API method. Returns the session's distinguished message listener, this is an expert facility not used by regular JMS clients.

Returns:

the message listener associated with this session

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

final int Session.getQueueMessageReadMax ()

Get the maximum number of messages that can be read at once from a queue for this Session.

This attribute is inherited from Connection at initialization, default value is 1.

Returns:

The maximum number of messages that can be read at once from a queue.

See also:

FactoryParameters::queueMessageReadMax

final int Session.getTopicAckBufferMax ()

Get the maximum number of acknowledgements that can be buffered when using Session.DUPS_OK_ACKNOWLEDGE mode for this session.
This attribute is inherited from Connection at initialization.

Returns:

The Maximum number of acknowledgements that can be buffered when using Session.DUPS_OK_ACKNOWLEDGE mode.

See also:

FactoryParameters::topicAckBufferMax

final int Session.getTopicActivationThreshold ()

Get the threshold of activation for this session.
This threshold is the minimum messages number below which the subscription is activated.
This attribute is inherited from Connection at initialization, default value is 0.

See also:

FactoryParameters::topicActivationThreshold

Returns:

The minimum messages number below which the subscription is activated.

final int Session.getTopicPassivationThreshold ()

Get the threshold of passivation for this session.
This threshold is the maximum messages number over which the subscription is passivated.
This attribute is inherited from Connection at initialization, default value is Integer.MAX_VALUE.

Returns:

The maximum messages number over which the subscription is passivated.

See also:

FactoryParameters::topicPassivationThreshold

synchronized final boolean Session.getTransacted () throws JMSException

API method. Indicates whether the session is in transacted mode.

Returns:

true if the session is in transacted mode

Exceptions:

IllegalStateException	If the session is closed.
-----------------------	---------------------------

boolean Session.isAsyncSend ()

Indicates whether the messages produced are asynchronously sent or not (without or with acknowledgment).
This attribute is inherited from Connection at initialization, by default false.

Returns:

true if messages produced are asynchronously sent.

See also:

FactoryParameters::asyncSend

boolean Session.isAsyncSub ()

Indicates whether the subscription request is asynchronously handled or not.

Default value is false, the subscription is handled synchronously so the topic must be accessible.

Returns:

true if the subscription requests are asynchronously handled.

Since:

JORM 5.0.7

boolean Session.isImplicitAck ()

Indicates whether the messages consumed are implicitly acknowledged or not. If true messages are immediately removed from queue when delivered.

This attribute is inherited from Connection at initialization, by default false.

Returns:

true if messages produced are implicitly acknowledged.

See also:

FactoryParameters::implicitAck

final int Connection.getCompressedMinSize()

Returns the minimum size beyond which a message body is compressed for this session.

This attribute is inherited from Connection at initialization, the default value is 0 (no compression).

Returns:

the minimum size beyond which the message body is compressed.

See also:

FactoryParameters::compressedMinSize

final void setCompressedMinSize(int compressedMinSize)

Sets the minimum size beyond which the message body is compressed for this session.

Parameters:

compressedMinSize	the minimum size beyond which the message body is compressed.
-------------------	---

See also:

FactoryParameters::compressedMinSize

final int Connection.getCompressionLevel()

Returns the compression level (0..9) used when a message body is compressed in this session.

This attribute is inherited from Connection at initialization, the default value is 1 (Deflater.BEST_SPEED).

Returns:

The compression level.

See also:

FactoryParameters::compressionLevel

final void setCompressionLevel(int compressionLevel)

Sets the compression level (0..9) used when a message body is compressed in this session.

This attribute is inherited from Connection at initialization, the default value is 1 (Deflater.BEST_SPEED).

Parameters:

compressedLevel	the compression level (0..9) used when a message body is compressed.
-----------------	--

See also:

FactoryParameters::compressionLevel

synchronized void Session.recover () throws JMSException

API method. Stops message delivery in this session, and restarts message delivery with the oldest unacknowledged message.

Exceptions:

IllegalStateException	If the session is closed, or transacted.
-----------------------	--

synchronized void Session.rollback () throws JMSException

API method. Rolls back any messages done in this transaction and releases any locks currently held.

Exceptions:

IllegalStateException	If the session is closed, or not transacted.
-----------------------	--

synchronized void Session.run ()

API method. Optional operation, intended to be used only by Application Servers, not by ordinary JMS clients.

void Session.setAsyncSend (boolean asyncSend)

Sets asynchronously sending for this session.

Determines whether the messages produced are asynchronously sent or not (without or with acknowledgement). This attribute is inherited from Connection at initialization, by default false.

Parameters:

asyncSend	If true sets asynchronous sending for this session.
-----------	---

See also:

FactoryParameters::asyncSend

void Session.setAsyncSub (boolean asyncSub)

Sets asynchronous subscription for this session.

Determines whether the subscription request is asynchronously handled or not.

Default value is false, the subscription is handled synchronously so the topic must be accessible.

Parameters:

asyncSub	If true sets asynchronous subscription for this session.
----------	--

Since:

JORM 5.0.7

void Session.setImplicitAck (boolean implicitAck)

Sets implicit acknowledge for this session.

Determines whether the messages produced are implicitly acknowledged or not. If set to true the messages are immediately removed from queue when delivered.

This attribute is inherited from Connection at initialization, by default false.

Parameters:

implicitAck	If true sets implicit acknowledge for this session.
-------------	---

See also:

FactoryParameters::implicitAck

synchronized void Session.setMessageListener (javax.jms.MessageListener listener) throws JMSException

API method. Sets the session's distinguished message listener, this is an expert facility not used by regular JMS clients.

When the distinguished message listener is set, no other form of message receipt in the session can be used; however, all forms of sending messages are still supported.

Parameters:

listener	the message listener to associate with this session.
----------	--

Exceptions:

JMSException	Actually never thrown.
--------------	------------------------

void setQueueMessageReadMax (int queueMessageReadMax)

Set the maximum number of messages that can be read at once from a queue for this Session.

This attribute is inherited from Connection at initialization, default value is 1.

Parameters:

queueMessageReadMax	The maximum number of messages that can be read at once from a queue.
---------------------	---

See also:

FactoryParameters::queueMessageReadMax

void Session.setTopicAckBufferMax (int topicAckBufferMax)

Set the maximum number of acknowledgements that can be buffered when using Session.DUPS_OK_ACKNOWLEDGE mode for this session.

This attribute is inherited from Connection at initialization.

Parameters:

topicAckBufferMax	The Maximum number of acknowledgements that can be buffered in Session.DUPS_OK_ACKNOWLEDGE mode.
-------------------	--

See also:

FactoryParameters::topicAckBufferMax

void org.objectweb.joram.client.jms.setTopicActivationThreshold (int topicActivationThreshold)

Set the threshold of activation for this session.

This threshold is the minimum messages number below which the subscription is activated.

This attribute is inherited from Connection at initialization, default value is 0.

Parameters:

topicActivationThreshold	The minimum messages number below which the subscription is activated.
--------------------------	--

See also:

FactoryParameters::topicActivationThreshold

void Session.setTopicPassivationThreshold (int topicPassivationThreshold)

Set the threshold of passivation for this session.

This threshold is the maximum messages number over which the subscription is passivated.

This attribute is inherited from Connection at initialization, default value is Integer.MAX_VALUE.

Parameters:

topicPassivationThreshold	The maximum messages number over which the subscription is passivated.
---------------------------	--

See also:

FactoryParameters::topicPassivationThreshold

synchronized void Session.unsubscribe (String name) throws JMSEException

API method. Unsubscribes a durable subscription that has been created by a client, this method deletes the state being maintained on behalf of the subscriber by the Joram server.

It is erroneous for a client to delete a durable subscription while there is an active MessageConsumer for the subscription, or while a consumed message is part of a pending transaction or has not been acknowledged in the session.

Parameters:

name	the name used to identify this subscription.
------	--

Exceptions:

IllegalStateException	If the session is closed or if the connection is broken.
InvalidDestinationException	If the subscription does not exist.
JMSEException	If the request fails for any other reason.

Member Data Documentation

static int AUTO_ACKNOWLEDGE

With this acknowledgment mode, the session automatically acknowledges a client's receipt of a message either when the session has successfully returned from a call to receive or when the message listener the session has called to process the message successfully returns.

static int CLIENT_ACKNOWLEDGE

With this acknowledgment mode, the client acknowledges explicitly consumed messages by calling the message's acknowledge method. As defined by the JMS specification acknowledging a consumed message automatically acknowledges the receipt of all messages that have been delivered by its session

static int DUPS_OK_ACKNOWLEDGE

This acknowledgment mode instructs the session to lazily acknowledge the delivery of messages.

static int SESSION_TRANSACTED

This value is returned from the method getAcknowledgeMode if the session is transacted.

static int INDIVIDUAL_ACKNOWLEDGE

With this acknowledgment mode, the client acknowledges a consumed message by calling the message's acknowledge method. Contrary to CLIENT_ACKNOWLEDGE mode this mode allows to acknowledge only the specified message.

4.8. MessageConsumer

Class org.objectweb.joram.client.jms.MessageConsumer
Implements javax.jms.MessageConsumer

Public Member Functions

```
synchronized void setMessageListener (javax.jms.MessageListener messageListener) throws JMSException
synchronized javax.jms.MessageListener getMessageListener () throws JMSException
final String getMessageSelector () throws JMSException
javax.jms.Message receive (long timeOut) throws JMSException
javax.jms.Message receive () throws JMSException
javax.jms.Message receiveNoWait () throws JMSException
void close () throws JMSException
```

Detailed Description

Implements the javax.jms.MessageConsumer interface.

A client uses a MessageConsumer object to receive messages from a destination. A MessageConsumer object is created by calling the createConsumer method on a session object. A message consumer is normally dedicated to a unique destination.

A message consumer can be created with a message selector. A message selector allows the client to restrict the messages delivered to the message consumer to those that match the selector.

A client may either synchronously receive a message consumer's messages or have the consumer asynchronously deliver them as they arrive:

- For synchronous receipt, a client can request the next message from the message consumer using one of its receive methods. There are several variations of receive that allow a client to poll or wait for the next message.
- For asynchronous delivery, a client must register a MessageListener object with a message consumer. As messages arrive at the message consumer, it delivers them by calling the MessageListener's onMessage method. It is a client programming error for a MessageListener to throw an exception.

It is a client programming error for a MessageListener to throw an exception.

Member Function Documentation

void MessageConsumer.close () throws JMSException

API method.

Closes the message consumer.

In order to free significant resources allocated on behalf of a MessageConsumer, clients should close them when they are not needed.

This call blocks until a receive or message listener in progress has completed. A blocked message consumer receive call returns null when this message consumer is closed.

Exceptions:

JMSException	if closing the consumer fails due to some internal error.
--------------	---

synchronized javax.jms.MessageListener MessageConsumer.getMessageListener () throws JMSException

API method.

Gets the message consumer's MessageListener.

Returns:

the listener for the message consumer, or null if no listener is set.

Exceptions:

IllegalStateException	If the consumer is closed.
-----------------------	----------------------------

final String MessageConsumer.getMessageSelector () throws JMSException

API method.

Gets this message consumer's message selector expression.

Returns:

this message consumer's message selector, or null if no message selector is set.

Exceptions:

IllegalStateException	If the consumer is closed.
-----------------------	----------------------------

javax.jms.Message MessageConsumer.receive (long timeOut) throws JMSException

API method.

Receives the next message that arrives before the specified timeout.

This call blocks until a message is available, the timeout expires, or this message consumer is closed. A timeout of zero never expires, and the call blocks indefinitely.

Parameters:

timeout	the timeout value (in milliseconds).
---------	--------------------------------------

Returns:

the next message available for this message consumer, or null if the timeout expires or this message consumer is concurrently closed.

Exceptions:

IllegalStateException	If the consumer is closed, or if the connection is broken.
JMSecurityException	If the requester is not a READER on the destination.
JMSException	If the request fails for any other reason.

javax.jms.Message MessageConsumer.receive () throws JMSException

API method.

Receives the next message produced for this message consumer, this call blocks indefinitely until a message is available or until this message consumer is closed.

If this receive is done within a transaction, the consumer retains the message until the transaction commits.

Returns:

the next message available for this message consumer, or null if this message consumer is concurrently closed.

Exceptions:

IllegalStateException	If the consumer is closed, or if the connection is broken.
JMSecurityException	If the requester is not a READER on the destination.
JMSException	If the request fails for any other reason.

javax.jms.Message MessageConsumer.receiveNoWait () throws JMSException

API method.

Receives the next message if one is immediately available.

Returns:

the next message available for this message consumer, or null if none is available.

Exceptions:

IllegalStateException	If the consumer is closed, or if the connection is broken.
JMSecurityException	If the requester is not a READER on the destination.
JMSException	If the request fails for any other reason.

synchronized void MessageConsumer.setMessageListener (javax.jms.MessageListener messageListener) throws JMSException

Sets the message consumer's MessageListener. API method.

This method must not be called if the connection the consumer belongs to is started, because the session would then be accessed by the thread calling this method and by the thread controlling asynchronous deliveries. This situation is clearly forbidden by the single threaded nature of sessions. Moreover, unsetting a message listener without stopping the connection may lead to the situation where asynchronous deliveries would arrive on the connection, the session or the consumer without being able to reach their target listener!

Parameters:

messageListener	the listener to which the messages are to be delivered.
-----------------	---

Exceptions:

IllegalStateException	If the consumer is closed, or if the connection is broken.
JMSEException	If the request fails for any other reason.

4.9. MessageProducer

Class org.objectweb.joram.client.jms.MessageProducer
implements javax.jms.MessageProducer

Public Member Functions

```
synchronized void setDisableMessageID (boolean value) throws JMSEException
synchronized void setDeliveryMode (int deliveryMode) throws JMSEException
synchronized void setPriority (int priority) throws JMSEException
synchronized void setTimeToLive (long timeToLive) throws JMSEException
synchronized void setDisableMessageTimestamp (boolean value) throws JMSEException
synchronized javax.jms.Destination getDestination () throws JMSEException
synchronized boolean getDisableMessageID () throws JMSEException
synchronized int getDeliveryMode () throws JMSEException
synchronized int getPriority () throws JMSEException
synchronized long getTimeToLive () throws JMSEException
synchronized boolean getDisableMessageTimestamp () throws JMSEException
synchronized void send (javax.jms.Message message) throws JMSEException
synchronized void send (javax.jms.Message message, int deliveryMode, int priority, long timeToLive) throws
JMSEException
synchronized void send (javax.jms.Destination dest, javax.jms.Message message) throws JMSEException
synchronized void send (javax.jms.Destination dest, javax.jms.Message message, int deliveryMode, int priority, long
timeToLive) throws JMSEException
synchronized void close () throws JMSEException
```

Detailed Description

Implements the javax.jms.MessageProducer interface.

A client uses a MessageProducer object to send messages to a destination. A MessageProducer object is created by calling the createProducer method on the session object. A message producer is normally dedicated to a unique destination.

A client also has the option of creating a message producer without supplying a unique destination. In this case, a destination must be provided with every send operation.

A client can specify a default delivery mode, priority, and time to live for messages sent by a message producer. It can also specify the delivery mode, priority, and time to live for each individual message.

Member Function Documentation

synchronized void MessageProducer.close () throws JMSEException

API method. Closes the message producer.

In order to free significant resources allocated on behalf of a MessageProducer, clients should close them when they are not needed.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

synchronized int MessageProducer.getDeliveryMode () throws JMSEException

API method. Gets the producer's default delivery mode, by default Message.DEFAULT_DELIVERY_MODE.

Returns:

the message delivery mode for this message producer.

Exceptions:

IllegalStateException	If the producer is closed.
-----------------------	----------------------------

synchronized javax.jms.Destination MessageProducer.getDestination () throws JMSEException

API method. Gets the destination associated with this MessageProducer.

Returns:

the destination associated with this MessageProducer.

Exceptions:

IllegalStateException	If the producer is closed.
-----------------------	----------------------------

synchronized boolean MessageProducer.getDisableMessageID () throws JMSEException

API method. Gets an indication of whether message IDs are disabled, always false.

Returns:

false.

Exceptions:

IllegalStateException	If the producer is closed.
-----------------------	----------------------------

synchronized boolean MessageProducer.getDisableMessageTimestamp () throws JMSEException

API method. Gets an indication of whether message timestamps are disabled.

Returns:

an indication of whether message timestamps are disabled.

Exceptions:

IllegalStateException	If the producer is closed.
-----------------------	----------------------------

synchronized int org.objectweb.joram.client.jms.MessageProducer.getPriority () throws JMSEException

API method. Gets the producer's default priority, by default Message.DEFAULT_PRIORITY.

Returns:

the message priority for this message producer.

Exceptions:

IllegalStateException	If the producer is closed.
-----------------------	----------------------------

synchronized long MessageProducer.getTimeToLive () throws JMSEException

API method. Gets the default duration in milliseconds that a produced message should be retained by the provider, by default Message.DEFAULT_TIME_TO_LIVE.

Returns:

the message time to live in milliseconds; zero is unlimited.

Exceptions:

IllegalStateException	If the producer is closed.
-----------------------	----------------------------

synchronized void MessageProducer.send (javax.jms.Message message) throws JMSEException

API method. Sends a message with the MessageProducer's default delivery parameters.

Parameters:

message	the message to send.
---------	----------------------

Exceptions:

UnsupportedOperationException	If the dest is unidentified.
IllegalStateException	If the producer is closed, or if the connection is broken.
JMSEException	If the request fails for any other reason.

synchronized void MessageProducer.send (javax.jms.Message message, int deliveryMode, int priority, long timeToLive) throws JMSEException

API method. Sends a message to the destination with given delivery parameters.

Parameters:

message	the message to send.
deliveryMode	the delivery mode to use.
priority	the priority for this message.
timeToLive	the message's lifetime in milliseconds.

Exceptions:

UnsupportedOperationException	If the dest is unidentified.
IllegalStateException	If the producer is closed, or if the connection is broken.
JMSEException	If the request fails for any other reason.

synchronized void MessageProducer.send (javax.jms.Destination dest, javax.jms.Message message) throws JMSEException

API method. Sends a message to a destination for an unidentified message producer using default delivery parameters.

Typically, a message producer is assigned a destination at creation time; however, the JMS API also supports unidentified message producers, which require that the destination be supplied every time a message is sent.

Parameters:

dest	the destination to send this message to.
message	the message to send.

Exceptions:

UnsupportedOperationException	When the producer did not properly identify itself.
JMSecurityException	If the user is not a WRITER on the specified destination.
IllegalStateException	If the producer is closed, or if the connection is broken.
JMSEException	If the request fails for any other reason.

synchronized void MessageProducer.send (javax.jms.Destination dest, javax.jms.Message message, int deliveryMode, int priority, long timeToLive) throws JMSEException

API method. Sends a message to a destination for an unidentified message producer with given delivery parameters.

Typically, a message producer is assigned a destination at creation time; however, the JMS API also supports unidentified message producers, which require that the destination be supplied every time a message is sent.

Parameters:

dest	the destination to send this message to.
message	the message to send.
deliveryMode	the delivery mode to use.
priority	the priority for this message.
timeToLive	the message's lifetime in milliseconds.

Exceptions:

UnsupportedOperationException	When the producer did not properly identify itself.
-------------------------------	---

JMSecurityException	If the user is not a WRITER on the specified destination.
IllegalStateException	If the producer is closed, or if the connection is broken.
JMSEException	If the request fails for any other reason.

synchronized void MessageProducer.setDeliveryMode (int deliveryMode) throws JMSEException

API method. Sets the producer's default delivery mode.

Delivery mode is set to PERSISTENT by default.

Parameters:

deliveryMode	the message delivery mode for this message producer; legal values are DeliveryMode.NON_PERSISTENT and DeliveryMode.PERSISTENT.
--------------	--

Exceptions:

IllegalStateException	If the producer is closed.
JMSEException	When setting an invalid delivery mode.

synchronized void MessageProducer.setDisableMessageID (boolean value) throws JMSEException

API method, not taken into account.

Message IDs are always enabled.

Parameters:

value	indicates if message IDs are disabled, not taken in account.
-------	--

Exceptions:

IllegalStateException	If the producer is closed.
-----------------------	----------------------------

synchronized void MessageProducer.setDisableMessageTimestamp (boolean value) throws JMSEException

API method. Sets whether message timestamps are disabled.

Since timestamps take some effort to create and increase a message's size, Joram optimizes message overhead if it is given a hint that the timestamp is not used by an application. By calling the setDisableMessageTimestamp method on this message producer, a JMS client enables this potential optimization for all messages sent by this message producer (the produced messages have the timestamp set to zero).

Message timestamps are enabled by default.

Parameters:

value	indicates if message timestamps are disabled.
-------	---

Exceptions:

IllegalStateException	If the producer is closed.
-----------------------	----------------------------

synchronized void MessageProducer.setPriority (int priority) throws JMSEException

API method. Sets the producer's default priority.

The JMS API defines ten levels of priority value, with 0 as the lowest priority and 9 as the highest. Clients should consider priorities 0-4 as gradations of normal priority and priorities 5-9 as gradations of expedited priority.

Priority is set to 4 by default (Message.DEFAULT_PRIORITY).

Parameters:

priority	the message priority for this message producer; must be a value between 0 and 9.
----------	--

Exceptions:

IllegalStateException	If the producer is closed.
JMSEException	When setting an invalid priority.

synchronized void MessageProducer.setTimeToLive (long timeToLive) throws JMSEException

API method. Sets the default duration of time in milliseconds that a produced message should be retained by the provider.

Time to live is set to zero by default (Message.DEFAULT_TIME_TO_LIVE).

Parameters:

timeToLive	the message time to live in milliseconds; zero is unlimited.
------------	--

Exceptions:

IllegalStateException	If the producer is closed.
-----------------------	----------------------------

4.10. QueueBrowser

Class. org.objectweb.joram.client.jms.QueueBrowser

Implements javax.jms.QueueBrowser

Public Member Functions

synchronized javax.jms.Queue getQueue () throws JMSEException

synchronized String getMessageSelector () throws JMSEException

synchronized Enumeration getEnumeration () throws JMSEException

synchronized void close () throws JMSEException

Detailed Description

Implements the javax.jms.QueueBrowser interface.

A client uses a QueueBrowser object to look at messages on a queue without removing them.

The getEnumeration method returns a java.util.Enumeration that is used to scan the queue's messages. It may be an enumeration of the entire content of a queue, or it may contain only the messages matching a message selector. A QueueBrowser can be created from either a Session or a QueueSession.

Member Function Documentation

synchronized void QueueBrowser.close () throws JMSEException

API method. Closes the QueueBrowser.

In order to free significant resources allocated on behalf of a QueueBrowser, clients should close them when they are not needed.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

synchronized Enumeration QueueBrowser.getEnumeration () throws JMSEException

API method. Gets an enumeration for browsing the current queue messages in the order they would be received.

Returns:

an enumeration for browsing the messages.

Exceptions:

IllegalStateException	If the browser is closed, or if the connection is broken.
JMSecurityException	If the client is not a READER on the queue.
JMSEException	If the request fails for any other reason.

synchronized String QueueBrowser.getMessageSelector () throws JMSEException

API method. Gets this queue browser's message selector expression.

Returns:

this queue browser's message selector, or null if no message selector exists for the message consumer.

Exceptions:

IllegalStateException	If the browser is closed.
-----------------------	---------------------------

synchronized javax.jms.Queue QueueBrowser.getQueue () throws JMSException

API method. Gets the queue associated with this queue browser.

Returns:

the queue.

Exceptions:

IllegalStateException	If the browser is closed.
-----------------------	---------------------------

4.11. Destination

Class org.objectweb.joram.client.jms.Destination

Implements javax.jms.destination

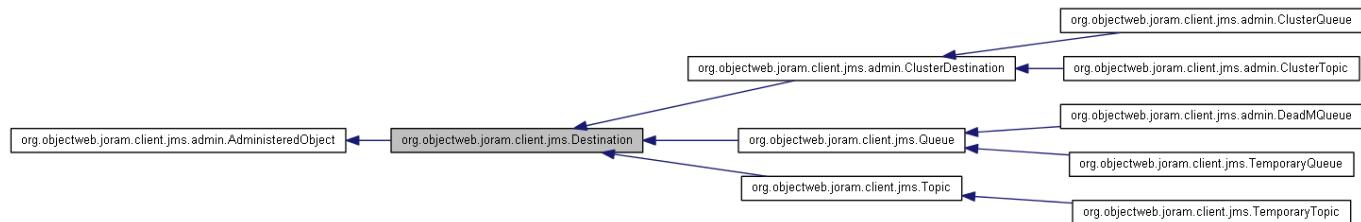


Figure 4.3 - Inheritance diagram for Destination class

Public Member Functions

```

String getName ()
byte getType ()
final String getAdminName ()
boolean equals (Object obj)
String toXml (int indent, int serverId) throws ConnectException, AdminException
boolean isQueue ()
boolean isTopic ()
void delete () throws ConnectException, AdminException, javax.jms.JMSException
void setFreeReading () throws ConnectException, AdminException
void setFreeWriting () throws ConnectException, AdminException
void unsetFreeReading () throws ConnectException, AdminException
void unsetFreeWriting () throws ConnectException, AdminException
void setReader (User user) throws ConnectException, AdminException
void setWriter (User user) throws ConnectException, AdminException
void unsetReader (User user) throws ConnectException, AdminException
void unsetWriter (User user) throws ConnectException, AdminException
List getReaders () throws ConnectException, AdminException
List getWriters () throws ConnectException, AdminException
boolean isFreelyReadable () throws ConnectException, AdminException
void setFreelyReadable (boolean b) throws ConnectException, AdminException
boolean isFreelyWriteable () throws ConnectException, AdminException
void setFreelyWriteable (boolean b) throws ConnectException, AdminException
Queue getDMQ () throws ConnectException, AdminException
String getDMQId () throws ConnectException, AdminException
void setDMQ (Queue dmq) throws ConnectException, AdminException, InvalidDestinationException
void setDMQId (String dmqid) throws ConnectException, AdminException, InvalidDestinationException
Hashtable getStatistic () throws ConnectException, AdminException
  
```

```
Hashtable getStatistics () throws ConnectException, AdminException
void addInterceptors (String interceptors) throws ConnectException, AdminException
String getInterceptors () throws ConnectException, AdminException
void removeInterceptors (String interceptors) throws ConnectException, AdminException
void replaceInterceptor (String newInterceptor, String oldInterceptor) throws ConnectException, AdminException
AdminReply setProperties (Properties prop) throws ConnectException, AdminException
```

Static Public Member Functions

static final void checkId (String id) throws InvalidDestinationException

Static Public Attributes

```
static final byte TOPIC_TYPE
static final byte QUEUE_TYPE
static final byte TEMPORARY
static final String QUEUE
static final String TOPIC
static final String DEAD_QUEUE
static final String CLUSTER_QUEUE
static final String SCHEDULER_QUEUE
static final String ACQUISITION_QUEUE
static final String DISTRIBUTION_QUEUE
static final String ACQUISITION_TOPIC
static final String DISTRIBUTION_TOPIC
```

Detailed Description

Implements the javax.jms.Destination interface and provides JORAM specific administration and monitoring methods.

A Destination is a JMS administered object that encapsulates a Joram's specific address. It is created by an administrator and later used by JMS clients. Normally the JMS clients find administered objects by looking them up in a JNDI namespace.

Joram MOM Model

Server side, a destination is a component receiving messages from producers and answering to consuming requests from consumers. A destination might either be a "queue" or a "topic":

- Queue: each message is read only by a single client.
- Topic: All clients that have previously subscribed to this topic are notified of the corresponding message.

A destination allows clients to perform operations according to their access rights. A client set as a READER will be able to request messages from the destination (either as a subscriber to a topic, or as a receiver or browser on a queue). A client set as a WRITER will be able to send messages to the destination.

A destination provides methods to add and remove Interceptors, an interceptor is an object handling each message sent to the destination. Interceptors can read and also modify the messages. This enables filtering, transformation or content enrichment, for example adding a property into the message. Also Interceptors can stop the Interceptor chain by simply returning false to their intercept method invocation, in this case the transmission of the message is stopped.

Member Function Documentation

void Destination.addInterceptors (String interceptors) throws ConnectException, AdminException

Administration method add interceptors.

Parameters:

interceptors	list of string className interceptor (separate with ",")
--------------	--

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static final Destination.checkId (String id) throws InvalidDestinationException [static]

Check the specified destination identifier.

Exceptions:

Exception	if an invalid destination identifier is specified.
-----------	--

void Destination.delete () throws ConnectException, AdminException, javax.jms.JMSEException

Administration method removing this destination from the platform.

Exceptions:

AdminException	Never thrown.
ConnectException	If the administration connection is closed or broken.
JMSEException	Never thrown.

boolean Destination.equals (Object obj)

Returns true if the parameter object is a Joram destination wrapping the same Joram's Destination.
final String Destination.getAdminName ()

Returns the symbolic administration name of the destination. This symbolic name is given by the user at creation, if it is unknown the internal name of this destination is returned..

Returns:

the symbolic name of the destination if any.

Queue Destination.getDMQ () throws ConnectException, AdminException

Monitoring method returning the dead message queue of this destination, null if not set.

The request fails if the destination is deleted server side.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

String Destination.getDMQId () throws ConnectException, AdminException

Monitoring method returning the dead message queue id of this destination, null if not set.

The request fails if the destination is deleted server side.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

String Destination.getInterceptors () throws ConnectException, AdminException

Administration method to get interceptors list.

Returns:

list of string className interceptor (separate with ",")

Exceptions:

ConnectException	If the administration connection is closed or broken.
------------------	---

AdminException	If the request fails.
----------------	-----------------------

String Destination.getName ()

Returns the internal name of the destination. This unique name is chosen internally by the MOM.

Returns:

the internal name of the destination.

List Destination.getReaders () throws ConnectException, AdminException

Monitoring method returning the list of all users that have a reading permission on this destination, or an empty list if no specific readers are set.

The request fails if the destination is deleted server side.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

Hashtable Destination.getStatistic () throws ConnectException, AdminException

Return a set of statistic values from the destination.

See also:

[org.objectweb.joram.client.jms.DestinationMBean::getStatistic\(\)](#)

Deprecated:**Hashtable Destination.getStatistics () throws ConnectException, AdminException**

Returns values of all valid JMX attributes about the destination.

Returns:

a Hashtable containing the values of all valid JMX attributes about the destination. The keys are the name of corresponding attributes.

See also:

[org.objectweb.joram.client.jms.DestinationMBean::getStatistics\(\)](#)

byte Destination.getType ()

Returns the type of the destination: queue or topic, temporary or not.

List Destination.getWriters () throws ConnectException, AdminException

Monitoring method returning the list of all users that have a writing permission on this destination, or an empty list if no specific writers are set.

The request fails if the destination is deleted server side.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

boolean Destination.isFreelyReadable () throws ConnectException, AdminException

Monitoring method returning true if this destination provides free READ access.

The request fails if the destination is deleted server side.

Exceptions:

ConnectException	If the administration connection is closed or broken.
------------------	---

AdminException	If the request fails.
----------------	-----------------------

boolean Destination.isFreelyWriteable () throws ConnectException, AdminException

Monitoring method returning true if this destination provides free WRITE access.
The request fails if the destination is deleted server side.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

boolean Destination.isQueue ()

Returns true if the destination is a queue.

boolean Destination.isTopic ()

Returns true if the destination is a topic.

void Destination.removeInterceptors (String interceptors) throws ConnectException, AdminException

Administration method to remove interceptors.

Parameters:

interceptors	list of string className interceptor (separate with ",")
--------------	--

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

void Destination.replaceInterceptor (String newInterceptor, String oldInterceptor) throws ConnectException, AdminException

Administration method to replace interceptor.

Parameters:

newInterceptor	the new className interceptor.
oldInterceptor	the old className interceptor.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

void Destination.setDMQ (Queue dmq) throws ConnectException, AdminException, InvalidDestinationException

Administration method setting or unsetting a dead message queue for this destination.
The request fails if this destination is deleted server side.

Parameters:

dmq	The dead message queue to be set (null for unsetting current DMQ).
-----	--

Exceptions:

IllegalArgumentException	If the DMQ is not a valid JORAM destination.
ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.
InvalidDestinationException	If the specified destination is invalid.

void Destination.setDMQId (String dmqId) throws ConnectException, AdminException, InvalidDestinationException

Administration method setting or unsetting a dead message queue for this destination.
The request fails if this destination is deleted server side.

Parameters:

dmqId	The dead message queue Id to be set (null for unsetting current DMQ).
-------	---

Exceptions:

IllegalArgumentException	If the DMQ is not a valid JORAM destination.
ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.
InvalidDestinationException	If the specified destination is invalid.

void Destination.setFreelyReadable (boolean b) throws ConnectException, AdminException

Administration method (un)setting free reading access to this destination.
This method should be only used by the JMX MBean.

Parameters:

readable	if true set the free reading access else disable.
----------	---

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

See also:

[org.objectweb.joram.client.jms.DestinationMBean::setFreelyReadable\(boolean\)](#)

void Destination.setFreelyWriteable (boolean b) throws ConnectException, AdminException

Administration method (un)setting free writing access to this destination.
This method should be only used by the JMX MBean.

Parameters:

writeable	if true set the free writing access else disable.
-----------	---

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

See also:

[org.objectweb.joram.client.jms.DestinationMBean::setFreelyWriteable\(boolean\)](#)

void Destination.setFreeReading () throws ConnectException, AdminException

Administration method setting free reading access to this destination.
The request fails if this destination is deleted server side.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

void Destination.setFreeWriting () throws ConnectException, AdminException

Administration method setting free writing access to this destination.
The request fails if this destination is deleted server side.

Exceptions:

ConnectException	If the administration connection is closed or broken.
------------------	---

n	
AdminException	If the request fails.

AdminReply Destination.setProperties (Properties prop) throws ConnectException, AdminException

Administration method to set properties.

Parameters:

prop	the properties to update.
------	---------------------------

Returns:

the admin reply

Exceptions:

ConnectException	
AdminException	

void Destination.setReader (User user) throws ConnectException, AdminException

Administration method setting a given user as a reader on this destination.

The request fails if this destination is deleted server side.

Parameters:

user	User to be set as a reader.
------	-----------------------------

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

void Destination.setWriter (User user) throws ConnectException, AdminException

Administration method setting a given user as a writer on this destination.

The request fails if this destination is deleted server side.

Parameters:

user	User to be set as a writer.
------	-----------------------------

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

String Destination.toXml (int indent, int serverId) throws ConnectException, AdminException

Format the destination properties in a XML format, the result can be used in an XML configuration script.

Parameters:

indent	use this indent for prefixing XML representation.
serverId	server id hosting the destination object

Returns:

returns a XML view of the queue (XML configuration format)

Exceptions:

ConnectException	if the server is unreachable
AdminException	if an error occurs

void Destination.unsetFreeReading () throws ConnectException, AdminException

Administration method unsetting free reading access to this destination.

The request fails if this destination is deleted server side.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

void Destination.unsetFreeWriting () throws ConnectException, AdminException

Administration method unsetting free writing access to this destination.

The request fails if this destination is deleted server side.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

void Destination.unsetReader (User user) throws ConnectException, AdminException

Administration method unsetting a given user as a reader on this destination.

The request fails if this destination is deleted server side.

Parameters:

user	Reader to be unset.
------	---------------------

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

void Destination.unsetWriter (User user) throws ConnectException, AdminException

Administration method unsetting a given user as a writer on this destination.

The request fails if this destination is deleted server side.

Parameters:

user	Writer to be unset.
------	---------------------

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

Member Data Documentation**final byte Destination.QUEUE_TYPE [static]**

Constant defining the type of a queue destination.

See also:

getType()

final byte Destination.TEMPORARY [static]

Constant defining the type of a temporary destination (OR'ed with queue or topic type depending of the real type of the destination).

See also:

getType()

final byte Destination.TOPIC_TYPE [static]

Constant defining the type of a topic destination.

See also:

`getType()`

final String Destination.QUEUE [static]

Constant defining the implementation class for a classic Queue.

final String Destination.TOPIC [static]

Constant defining the implementation class for a classic Topic.

final String Destination.DEAD_MQUEUE = "org.objectweb.joram.mom.dest.Queue"; [static]

Constant defining the implementation class for a Dead Message Queue.

Deprecated:

Since Joram 5.2.2 the DeadMQueue is a simple Queue.

final String Destination.CLUSTER_QUEUE [static]

Constant defining the implementation class for a clustered Queue.

final String Destination.SCHEDULER_QUEUE [static]

Constant defining the implementation class for a scheduled Queue.

final String Destination.ACQUISITION_QUEUE [static]

Constant defining the implementation class for a Queue allowing to collect data from external sources. The nature of data collector is configurable through properties.

final String Destination.DISTRIBUTION_QUEUE [static]

Constant defining the implementation class for a Queue allowing to forward data to external targets. The nature of data forwarder is configurable through properties.

final String Destination.ACQUISITION_TOPIC [static]

Constant defining the implementation class for a Topic allowing to collect data from external sources. The nature of data collector is configurable through properties.

final String Destination.DISTRIBUTION_TOPIC [static]

Constant defining the implementation class for a Queue allowing to forward data to external targets. The nature of data forwarder is configurable through properties.

4.12. Queue

Class `org.objectweb.joram.client.jms.Queue`

Extends `Destination`

Implements `javax.jms.Queue`

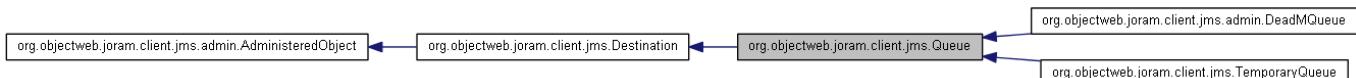


Figure 4.4 - Inheritance diagram for Queue class

Public Member Functions

```

String getQueueName () throws JMSException
void setThreshold (int threshold) throws ConnectException, AdminException
int getThreshold () throws ConnectException, AdminException
void setNbMaxMsg (int nbMaxMsg) throws ConnectException, AdminException
int getNbMaxMsg () throws ConnectException, AdminException
void setSyncExceptionOnFull (boolean syncExceptionOnFull) throws ConnectException, AdminException
int getPendingMessages () throws ConnectException, AdminException
int getPendingRequests () throws ConnectException, AdminException
int getDeliveredMessages () throws ConnectException, AdminException
String[] getMessageIds () throws AdminException, ConnectException
javax.jms.Message getMessage (String msgId) throws AdminException, ConnectException, JMSException
void addClusteredQueue (Queue addedQueue) throws ConnectException, AdminException
void removeClusteredQueue (Queue removedQueue) throws ConnectException, AdminException
void removeFromCluster () throws ConnectException, AdminException
String[] getQueueClusterElements () throws ConnectException, AdminException
void registerAsDefaultDMQ () throws ConnectException, AdminException
void registerAsDefaultDMQ (int serverId) throws ConnectException, AdminException
void addRemoteDestination (String newId) throws ConnectException, AdminException
void delRemoteDestination (String newId) throws ConnectException, AdminException
void setDeliveryDelay(int deliveryDelay) throws ConnectException, AdminException
void setRedeliveryDelay(int redeliveryDelay) throws ConnectException, AdminException
void setPause(boolean pause) throws ConnectException, AdminException
  
```

Static Public Member Functions

```

static Queue create () throws ConnectException, AdminException
static Queue create (int serverId) throws ConnectException, AdminException
static Queue create (String name) throws ConnectException, AdminException
static Queue create (int serverId, String name) throws ConnectException, AdminException
static Queue create (int serverId, Properties prop) throws ConnectException, AdminException
static Queue create (int serverId, String className, Properties prop) throws ConnectException, AdminException
static Queue create (int serverId, String name, String className, Properties prop) throws ConnectException, AdminException
  
```

Detailed Description

Implements the `javax.jms.Queue` interface.
 This is a proxy object a client uses to specify the destination of messages it is sending and the source of messages it receives.
 The Queue class is a factory for Joram's Queue destination through the create static methods, the Queue object provides Joram specific administration and monitoring methods.

Member Function Documentation

void Queue.addClusteredQueue (Queue addedQueue) throws ConnectException, AdminException

Adds a queue into the cluster this queue belongs to. If this queue doesn't belong to a cluster then a cluster is created by clustering this queue with the added queue.

The request fails if one or both of the queues are deleted, or can't belong to a cluster.

Parameters:

addedQueue	queue added to the cluster
------------	----------------------------

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

void Queue.addRemoteDestination (String newId) throws ConnectException, AdminException

Adds a destination to an alias queue's destinations' list.

Parameters:

destId	
--------	--

Exceptions:

ConnectException	
AdminException	

static Queue Queue.create () throws ConnectException, AdminException [static]

Admin method creating and deploying a queue on the local server.

The request fails if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

static Queue Queue.create (int serverId) throws ConnectException, AdminException [static]

Admin method creating and deploying a queue on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
----------	---

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

static Queue org.objectweb.joram.client.jms.Queue.create (String name) throws ConnectException, AdminException [static]

Admin method creating and deploying (or retrieving) a queue on the local server. First a destination with the specified name is searched on the given server, if it does not exist it is created. In any case, its provider-specific address is returned.

The request fails if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

name	The queue name.
------	-----------------

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

static Queue Queue.create (int serverId, String name) throws ConnectException, AdminException [static]

Admin method creating and deploying (or retrieving) a queue on a given server with a given name. First a destination with the specified name is searched on the given server, if it does not exist it is created. In any case, its provider-specific address is returned.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The queue name.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

static Queue Queue.create (int serverId, Properties prop) throws ConnectException, AdminException [static]

Admin method creating and deploying a queue on a given server. It creates a Joram's standard queue.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Properties:

deliveryDelay	The delivery delay in milliseconds used to wait before delivering a message. If set the resulting delay is the max between this value and the message property.
redeliveryDelay	The re-delivery delay use to wait before re-delivering messages after a deny.
period	The property defining the period of periodic tasks.
fixedInMemory	If true the created destination is pinned in memory. As a side effect the destination is preloaded during the server initialization.
jms_joram_interceptors	The property defining the list of interceptors.

Parameters:

serverId	The identifier of the server where deploying the queue.
prop	The queue properties.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

static Queue Queue.create (int serverId, String className, Properties prop) throws ConnectException, AdminException [static]

Admin method creating and deploying a queue on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Properties:

deliveryDelay	The delivery delay in milliseconds used to wait before delivering a message. If set the resulting delay is the max between this value and the message property.
redeliveryDelay	The re-delivery delay use to wait before re-delivering messages after a deny.
period	The property defining the period of periodic tasks.

jms_joram_interceptors	The property defining the list of interceptors.
------------------------	---

Parameters:

serverId	The identifier of the server where deploying the queue.
className	The queue class name.
prop	The queue properties.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

static Queue Queue.create (int serverId, String name, String className, Properties prop) throws ConnectException, AdminException [static]

Admin method creating and deploying (or retrieving) a queue on a given server. First a destination with the specified name is searched on the given server, if it does not exist it is created. In any case, its provider-specific address is returned.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the queue.
className	The MOM's queue class name.
prop	The queue properties.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

void Queue.delRemoteDestination (String newId) throws ConnectException, AdminException

Removes a destination from an alias queue's destinations' list.

Parameters:

destId	
--------	--

Exceptions:

ConnectException	
AdminException	

int Queue.getDeliveredMessages () throws ConnectException, AdminException

Monitoring method returning the number of delivered messages since the queue's creation.

The request fails if the queue is deleted server side.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

javax.jms.Message Queue.getMessage (String msgId) throws AdminException, ConnectException, JMSException

Returns a copy of the message.

Parameters:

msgId	The identifier of the message.
-------	--------------------------------

Returns:

The message

Exceptions:

AdminException	
ConnectException	
JMSEException	

String [] Queue.getMessageIds () throws AdminException, ConnectException

Returns the identifiers of all messages in this queue.

Returns:

The identifiers of all messages in this queue.

See also:

[org.objectweb.joram.client.jms.QueueMBean::getMessageIds\(\)](#)

int Queue.getNbMaxMsg () throws ConnectException, AdminException

Monitoring method returning the nbMaxMsg of this queue, -1 if no limit.

The request fails if the queue is deleted server side.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

int Queue.getPendingMessages () throws ConnectException, AdminException

Monitoring method returning the number of pending messages on this queue.

The request fails if the queue is deleted server side.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

int Queue.getPendingRequests () throws ConnectException, AdminException

Monitoring method returning the number of pending requests on this queue.

The request fails if the queue is deleted server side.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

String [] Queue.getQueueClusterElements () throws ConnectException, AdminException

Returns the reference of the queues that belong to the cluster.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

String Queue.getQueueName () throws JMSEException

API method. Gets the The Joram's internal unique identifier of this queue.

Returns:

The Joram's internal unique identifier.

Exceptions:

JMSException	Actually never thrown.
--------------	------------------------

int Queue.getThreshold () throws ConnectException, AdminException

Monitoring method returning the threshold of this queue, -1 if not set.

The request fails if the queue is deleted server side.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

javax.jms.Message Queue.readMessage (String msgId) throws AdminException, ConnectException, JMSException

Returns a copy of the message.

Parameters:

msgId	The identifier of the message.
-------	--------------------------------

Returns:

The message

Exceptions:

AdminException	
ConnectException	
JMSException	

Deprecated:

Since Joram 5.2 use getMessage.

void Queue.registerAsDefaultDMQ () throws ConnectException, AdminException

Sets the current queue as the default DMQ for the local server.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

void Queue.registerAsDefaultDMQ (int serverId) throws ConnectException, AdminException

Sets the current queue as the default DMQ for the given server.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	The identifier of the server.
----------	-------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void Queue.removeClusteredQueue (Queue removedQueue) throws ConnectException, AdminException

Removes a queue from the cluster this queue belongs to.

The request fails if the queue does not exist or is not part of any cluster.

Parameters:

removedQueue	queue removed from the cluster
--------------	--------------------------------

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

Deprecated:**void Queue.removeFromCluster () throws ConnectException, AdminException**

Removes this queue from the cluster it belongs to.

The request fails if the queue does not exist or is not part of any cluster.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

void Queue.setNbMaxMsg (int nbMaxMsg) throws ConnectException, AdminException

Admin method setting nbMaxMsg for this queue.

The request fails if the queue is deleted server side.

Parameters:

nbMaxMsg	nb Max of Message (-1 no limit).
----------	----------------------------------

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

void Queue.setSyncExceptionOnFull (boolean syncExceptionOnFull) throws ConnectException, AdminException

Administration method setting syncExceptionOnFull for this queue.

The request fails if the queue is deleted server side.

Parameters:

syncExceptionOnFull	true, throws an exception on sending message on full destination.
---------------------	---

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

void Queue.setThreshold (int threshold) throws ConnectException, AdminException

Administration method setting (default not set) or unsetting the threshold for this queue. This property fix the value which messages are considered as undeliverable because constantly denied.

The request fails if the queue is deleted server side.

Parameters:

threshold	The threshold value to be set (-1 for unsetting previous value).
-----------	--

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

void Queue.setDeliveryDelay (int deviveryDelay) throws ConnectException, AdminException

Administration method setting the minimum delivery delay for this queue. The effective delay is the maximum between this value and the corresponding message property.

The request fails if the queue is deleted server side.

Parameters:

deliveryDelay	The minimum delay in milliseconds use to wait before delivering each message.
---------------	---

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

void Queue.setRedeliveryDelay (int redeviveryDelay) throws ConnectException, AdminException

Administration method setting the redelivery delay for this queue. The redelivery delay is the minimum delay that the queue wait to redeliver a message after a client denies this message.

The request fails if the queue is deleted server side.

Parameters:

redeliveryDelay	The delay in seconds use to wait before re-delivering messages after a deny.
-----------------	--

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

void Queue.setPause (boolean pause) throws ConnectException, AdminException

Administration method stopping (resp. resuming) the message delivery.

The request fails if the queue is deleted server side.

Parameters:

pause	If true stops the message delivery, else resumes it.
-------	--

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

4.13. TemporaryQueue

Class org.objectweb.joram.client.jms.TemporaryQueue

Extends Queue

Implements javax.jms.TemporaryQueue

Public Member Functions

void delete () throws JMSEException

Detailed Description

Implements the javax.jms.TemporaryQueue interface.

A TemporaryQueue object is a Queue object created for the duration of a Connection. It is a system-defined queue that can be consumed only by the Connection that created it. A TemporaryQueue object can be created at either the Session or QueueSession level.

Member Function Documentation

void TemporaryQueue.delete () throws JMSEException

API method. Deletes this temporary queue. If there are existing receivers still using it, a JMSEException will be thrown.

Exceptions:

IllegalStateException	If the connection is closed or broken.
JMSEException	If the request fails for any other reason.

4.14. Topic

Class org.objectweb.joram.client.jms.Topic

Extends Destination

Implements javax.jms.Topic

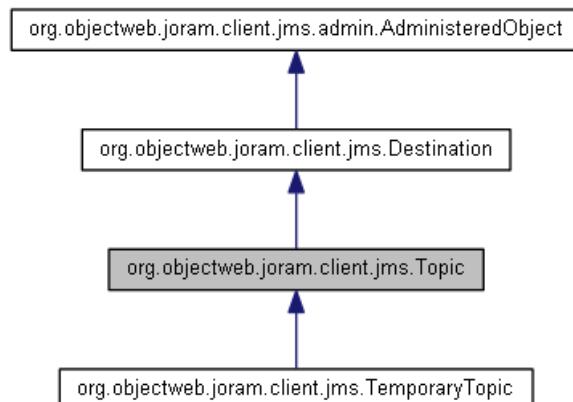


Figure 4.5 - Inheritance diagram for Topic class

Public Member Functions

```

String getTopicName () throws JMSEException
Topic getHierarchicalFather () throws ConnectException, AdminException
List getClusterFellows () throws ConnectException, AdminException
int getSubscriptions () throws ConnectException, AdminException
void addClusteredTopic (Topic addedTopic) throws ConnectException, AdminException
void removeFromCluster () throws ConnectException, AdminException
void setParent (Topic parent) throws ConnectException, AdminException
void unsetParent () throws ConnectException, AdminException
  
```

Static Public Member Functions

```

static Topic create (int serverId, String name, String className, Properties prop) throws ConnectException,
AdminException
static Topic create (int serverId, String className, Properties prop) throws ConnectException, AdminException
static Topic create (int serverId, Properties prop) throws ConnectException, AdminException
static Topic create (int serverId, String name) throws ConnectException, AdminException
  
```

```
static Topic create (String name) throws ConnectException, AdminException
static Topic create (int serverId) throws ConnectException, AdminException
static Topic create () throws ConnectException, AdminException
```

Detailed Description

Implements the javax.jms.Topic interface.

This is a proxy object a client uses to specify the destination of messages it is sending and the source of messages it receives.

The Topic class is a factory for Joram's Topic destination through the create static methods, the Topic object provides Joram specific administration and monitoring methods.

Member Function Documentation

void Topic.addClusteredTopic (Topic addedTopic) throws ConnectException, AdminException

Adds a topic into the cluster this topic belongs to. If this topic doesn't belong to a cluster then a cluster is created by clustering this topic with the added topic.

The request fails if one or both of the topics are deleted, or can't belong to a cluster.

Parameters:

addedTopic	topic added to the cluster
------------	----------------------------

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

static Topic create (int serverId, String name, String className, Properties prop) throws ConnectException, AdminException [static]

Admin method creating and deploying (or retrieving) a topic on a given server. First a destination with the specified name is searched on the given server, if it does not exist it is created. In any case, its provider-specific address is returned.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the topic.
className	The topic class name.
prop	The topic properties.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

static Topic Topic.create (int serverId, String className, Properties prop) throws ConnectException, AdminException [static]

Admin method creating and deploying a topic on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
className	The topic class name.

prop	The topic properties.
------	-----------------------

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

static Topic Topic.create (int serverId, Properties prop) throws ConnectException, AdminException [static]

Admin method creating and deploying a topic on a given server. It creates a Joram's standard topic. The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
prop	The topic properties.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

static Topic Topic.create (int serverId, String name) throws ConnectException, AdminException [static]

Admin method creating and deploying (or retrieving) a topic on a given server with a given name. First a destination with the specified name is searched on the given server, if it does not exist it is created. In any case, its provider-specific address is returned.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The topic name.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

static Topic Topic.create (String name) throws ConnectException, AdminException [static]

Admin method creating and deploying (or retrieving) a topic on the local server. First a destination with the specified name is searched on the given server, if it does not exist it is created. In any case, its provider-specific address is returned.

The request fails if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

name	The topic name.
------	-----------------

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

static Topic org.objectweb.joram.client.jms.Topic.create (int serverId) throws ConnectException, AdminException [static]

Admin method creating and deploying a topic on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
----------	---

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

static Topic org.objectweb.joram.client.jms.Topic.create () throws ConnectException, AdminException [static]

Admin method creating and deploying a topic on the local server.

The request fails if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

List Topic.getClusterFellows () throws ConnectException, AdminException

Monitoring method returning the list describing the cluster this topic is part of.

The request fails if the topic is deleted server side.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

Topic Topic.getHierarchicalFather () throws ConnectException, AdminException

Monitoring method returning the hierarchical father of this topic, null if none.

The request fails if the topic is deleted server side.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

int Topic.getSubscriptions () throws ConnectException, AdminException

Monitoring method returning the number of users that subscribes on this topic. If a client has many subscriptions it is only counted once.

The request fails if the topic is deleted server side.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

String Topic.getTopicName () throws JMSEException

API method. Gets the The Joram's internal unique identifier of this topic.

Returns:

The Joram's internal unique identifier.

Exceptions:

JMSException	Actually never thrown.
--------------	------------------------

void Topic.removeFromCluster () throws ConnectException, AdminException

Removes this topic from the cluster it belongs to.

The request fails if the topic does not exist or is not part of any cluster.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

void Topic.setParent (Topic parent) throws ConnectException, AdminException

Creates a hierarchical relationship between this topic and its father topic.

The request fails if one of the topics does not exist or can't be part of a hierarchy.

Parameters:

parent	the topic which will be parent. null to remove previous parent.
--------	---

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

void Topic.unsetParent () throws ConnectException, AdminException

Unsets the father of this topic.

The request fails if the topic does not exist or is not part of any hierarchy.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

4.15. TemporaryTopic

Class org.objectweb.joram.client.jms.TemporaryTopic

Extends Topic

Implements javax.jms.TemporaryTopic

Public Member Functions

void delete () throws JMSException

Detailed Description

Implements the javax.jms.TemporaryTopic interface.

A TemporaryTopic object is a Topic object created for the duration of a Connection. It is a system-defined topic that can be consumed only by the Connection that created it. A TemporaryTopic object can be created either at the Session or TopicSession level.

Member Function Documentation

void TemporaryTopic.delete () throws JMSException

API method. Deletes this temporary topic. If there are existing subscribers still using it, a JMSException will be thrown.

Exceptions:

IllegalStateException	If the connection is closed or broken.
JMSException	If the request fails for any other reason.

4.16. User

Class org.objectweb.joram.client.jms.admin.User

Public Member Functions

```

String getName ()
void update (String newName, String newPassword) throws ConnectException, AdminException
void update (String newName, String newPassword, String identityClassName) throws ConnectException, AdminException
void delete () throws ConnectException, AdminException
void setDMQ (Queue dmq) throws ConnectException, AdminException
void setDMQId (String dmqid) throws ConnectException, AdminException
Queue getDMQ () throws ConnectException, AdminException
String getDMQId () throws ConnectException, AdminException
void setThreshold (int threshold) throws ConnectException, AdminException
void setThreshold (String subname, int threshold) throws ConnectException, AdminException
int getThreshold () throws ConnectException, AdminException
int getThreshold (String subname) throws ConnectException, AdminException
void setNbMaxMsg (String subName, int nbMaxMsg) throws ConnectException, AdminException
int getNbMaxMsg (String subName) throws ConnectException, AdminException
Subscription[] getSubscriptions () throws AdminException, ConnectException
String[] getMessageIds (String subName) throws AdminException, ConnectException, JMSException
javax.jms.Message getMessage (String subName, String msgId) throws AdminException, ConnectException, JMSException
List getSubscriptionList () throws AdminException, ConnectException
Subscription getSubscription (String subName) throws AdminException, ConnectException
void addInterceptorsIN (String interceptors) throws ConnectException, AdminException
String getInterceptorsIN () throws ConnectException, AdminException
void removeInterceptorsIN (String interceptors) throws ConnectException, AdminException
void addInterceptorsOUT (String interceptors) throws ConnectException, AdminException
String getInterceptorsOUT () throws ConnectException, AdminException
void removeInterceptorsOUT (String interceptors) throws ConnectException, AdminException
void replaceInterceptorIN (String newInterceptor, String oldInterceptor) throws ConnectException, AdminException
void replaceInterceptorOUT (String newInterceptor, String oldInterceptor) throws ConnectException, AdminException
String getProxyId ()

```

Static Public Member Functions

```

static User create (String name, String password, int serverId) throws ConnectException, AdminException
static User create (String name, String password) throws ConnectException, AdminException
static User create (String name, String password, int serverId, String identityClassName) throws ConnectException, AdminException
static User create (String name, String password, int serverId, String identityClassName, Properties prop) throws ConnectException, AdminException

```

Detailed Description

The User class is a utility class needed for administering Joram users.

The User class is a factory for Joram's users through the create static methods, the User object provides Joram specific administration and monitoring methods.

The User object provides methods to add and remove Interceptors, such an interceptor can handle each incoming and outgoing message. Interceptors can read and also modify the messages. This enables filtering, transformation or content enrichment, for example adding a property into the message. Also Interceptors can stop the Interceptor chain by simply returning false to their intercept method invocation, in this case the transmission of the message is stopped.

There are two distinct chains of interceptors:

- The first one "interceptors_in" handles each message that's entering the server (result of a send method on a connection from the selected user).
- The second one "interceptors_out" handles each message that's exiting the server (result of a receive method on a connection from the selected user).

These two interceptor chains are configurable for each user.

Member Function Documentation

void User.addInterceptorsIN (String interceptors) throws ConnectException, AdminException

Add interceptors

Parameters:

interceptors	list of string className interceptor (separate with ",")
--------------	--

Exceptions:

ConnectException	
AdminException	

void User.addInterceptorsOUT (String interceptors) throws ConnectException, AdminException

Add interceptors

Parameters:

interceptors	list of string className interceptor (separate with ",")
--------------	--

Exceptions:

ConnectException	
AdminException	

static User User.create (String name, String password, int serverId) throws ConnectException, AdminException [static]

Admin method creating a user for a given server and instantiating the corresponding User object.

If the user has already been set on this server, the method simply returns the corresponding User object. It fails if the target server does not belong to the platform, or if a proxy could not be deployed server side for a new user.

Be careful this method uses the static AdminModule connection.

Parameters:

name	Name of the user.
password	Password of the user.
serverId	The identifier of the user's server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

static User User.create (String name, String password) throws ConnectException, AdminException [static]

Admin method creating a user on the local server and instantiating the corresponding User object.
If the user has already been set on this server, the method simply returns the corresponding User object. It fails if a proxy could not be deployed server side for a new user.
Be careful this method use the static AdminModule connection.

Parameters:

name	Name of the user.
password	Password of the user.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

static User User.create (String name, String password, int serverId, String identityClassName) throws ConnectException, AdminException [static]

Admin method creating a user for a given server and instantiating the corresponding User object.
If the user has already been set on this server, the method simply returns the corresponding User object. Its fails if the target server does not belong to the platform, or if a proxy could not be deployed server side for a new user.
Be careful this method use the static AdminModule connection.

Parameters:

name	Name of the user.
password	Password of the user.
serverId	The identifier of the user's server.
identityClassName	user/password or JAAS... (default SimpleIdentity).

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

static User User.create (String name, String password, int serverId, String identityClassName, Properties prop) throws ConnectException, AdminException [static]

Admin method creating a user for a given server and instantiating the corresponding User object.
If the user has already been set on this server, the method simply returns the corresponding User object. Its fails if the target server does not belong to the platform, or if a proxy could not be deployed server side for a new user.
Be careful this method use the static AdminModule connection.

Parameters:

name	Name of the user.
password	Password of the user.
serverId	The identifier of the user's server.
identityClassName	user/password or JAAS... (default SimpleIdentity).
prop	properties

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void User.delete () throws ConnectException, AdminException

Removes this user.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

boolean User.equals (Object o)

Returns true if the parameter object is a Joram user wrapping the same Joram's User.
 Queue User.getDMQ () throws ConnectException, AdminException
 Monitoring method returning the dead message queue of this user, null if not set.
 The request fails if the user is deleted server side.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

String User.getDMQId () throws ConnectException, AdminException

Monitoring method returning the dead message queue id of this user, null if not set.
 The request fails if the destination is deleted server side.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

String User.getInterceptorsIN () throws ConnectException, AdminException

Get interceptors.

Returns:

list of string className interceptor (separate with ",")

Exceptions:

ConnectException	
AdminException	

String User.getInterceptorsOUT () throws ConnectException, AdminException

Get interceptors.

Returns:

list of string className interceptor (separate with ",")

Exceptions:

ConnectException	
AdminException	

String User.getName ()

Returns the user name.

int User.getNbMaxMsg (String subName) throws ConnectException, AdminException

Monitoring method returning the nbMaxMsg of this subscription, -1 if no limit.
 The request fails if the sub is deleted server side.

Parameters:

subName	the name of the subscription.
---------	-------------------------------

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

String User.getProxyId ()

Returns the identifier of the user's proxy.

Subscription User.getSubscription (String subName) throws AdminException, ConnectException

Returns a subscription.

Parameters:

subName	the name of the subscription.
---------	-------------------------------

Returns:

The subscription.

Exceptions:

AdminException	If an error is raised by the administration operation.
ConnectException	If the admin connection is not established.

List User.getSubscriptionList () throws AdminException, ConnectException

used by MBean jmx

Subscription [] User.getSubscriptions () throws AdminException, ConnectException

Returns the subscriptions owned by a user.

Returns:

The subscriptions owned by the user.

Exceptions:

AdminException	If an error is raised by the administration operation.
ConnectException	If the admin connection is not established.

String [] Queue.getMessageIds (String subName) throws AdminException, ConnectException

Returns the identifiers of all messages in this queue.

Parameters:

SubName	The name of the related subscription.
---------	---------------------------------------

Returns:

The identifiers of all messages in this subscription.

Exceptions:

AdminException	
ConnectException	

javax.jms.Message Queue.getMessage (String subName, String msgId) throws AdminException, ConnectException, JMSException

Returns a copy of a message of the subscription.

Parameters:

SubName	The name of the related subscription.
---------	---------------------------------------

msgId	The identifier of the message.
-------	--------------------------------

Returns:

The message

Exceptions:

AdminException	If the request fails.
ConnectException	If the connection fails.
JMSEException	

int User.getThreshold () throws ConnectException, AdminException

Returns the threshold for this user, -1 if not set.

The request fails if the user is deleted server side.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

int User.getThreshold (String subname) throws ConnectException, AdminException

Returns the threshold for a particular subscription of this user, -1 if not set.

The request fails if the user is deleted server side.

Parameters:

subname	The subscription name.
---------	------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void User.removeInterceptorsIN (String interceptors) throws ConnectException, AdminException

Remove interceptors

Parameters:

interceptors	list of string className interceptor (separate with ",")
--------------	--

Exceptions:

ConnectException	
AdminException	

void User.removeInterceptorsOUT (String interceptors) throws ConnectException, AdminException

Remove interceptors

Parameters:

interceptors	list of string className interceptor (separate with ",")
--------------	--

Exceptions:

ConnectException	
AdminException	

void User.replaceInterceptorIN (String newInterceptor, String oldInterceptor) throws ConnectException, AdminException

Replace interceptor IN

Parameters:

newInterceptor	the new className interceptor.
oldInterceptor	the old className interceptor.

Exceptions:

ConnectException	
AdminException	

void User.replaceInterceptorOUT (String newInterceptor, String oldInterceptor) throws ConnectException, AdminException

Replace interceptor OUT

Parameters:

newInterceptor	the new className interceptor.
oldInterceptor	the old className interceptor.

Exceptions:

ConnectException	
AdminException	

void User.setDMQ (Queue dmq) throws ConnectException, AdminException

Admin method setting a given dead message queue for this user.

The request fails if the user is deleted server side.

Parameters:

dmq	The dead message queue to be set.
-----	-----------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void User.setDMQId (String dmqid) throws ConnectException, AdminException

Admin method setting a given dead message queue for this user.

The request fails if the user is deleted server side.

Parameters:

dmqid	The dead message queue Id to be set.
-------	--------------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void User.setNbMaxMsg (String subName, int nbMaxMsg) throws ConnectException, AdminException

Admin method setting nbMaxMsg for this subscription.

The request fails if the sub is deleted server side.

Parameters:

subName	the name of the subscription.
nbMaxMsg	nb Max of Message (-1 no limit).

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

void User.setThreshold (int threshold) throws ConnectException, AdminException

Administration method setting (default not set) or unsetting the threshold for this user. This property fix the value which messages are considered as undeliverable because constantly denied.
The request fails if the user is deleted server side.

Parameters:

threshold	The threshold value to be set (-1 for unsetting previous value).
-----------	--

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void User.setThreshold (String subname, int threshold) throws ConnectException, AdminException

Administration method setting (default not set) or unsetting the threshold for a particular subscription of this user. This property fix the value which messages are considered as undeliverable because constantly denied.
The request fails if the user is deleted server side.

Parameters:

subname	The subscription name.
threshold	The threshold value to be set (-1 for unsetting previous value).

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void User.update (String newName, String newPassword) throws ConnectException, AdminException

Admin method updating this user identification.

The request fails if the user does not exist server side, or if the new identification is already taken by a user on the same server.

Parameters:

newName	The new name of the user.
newPassword	The new password of the user.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void User.update (String newName, String newPassword, String identityClassName) throws ConnectException, AdminException

Admin method updating this user identification.

The request fails if the user does not exist server side, or if the new identification is already taken by a user on the same server.

Parameters:

newName	The new name of the user.
newPassword	The new password of the user.
identityClassName	user/password or JAAS... (default SimpleIdentity).

me	
----	--

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

4.17. Subscription

Class org.objectweb.joram.client.jms.admin.Subscription

Public Member Functions

```
final String getName ()
final String getTopicId ()
final int getMessageCount ()
final int getDeliveredMessageCount ()
final boolean isDurable ()
```

Detailed Description

The Subscription class is a utility class needed to show information about client subscription. Be careful, contrary to user, queue or topic administration object, the Subscription object is just a data structure; it is initialized by user.getSubscription() methods and no longer keep consistent with the real state of subscription.

Member Function Documentation

final int Subscription.getDeliveredMessageCount ()

Returns the number of messages delivered and waiting for acknowledge.

Returns:

The number of messages delivered and waiting for acknowledge.

final int Subscription.getMessageCount ()

Returns the number of pending messages.

Returns:

the number of pending messages.

final String Subscription.getName ()

Returns the subscription's name.

Returns:

the subscription's name.

final String Subscription.getTopicId ()

Returns the related topic unique identification.

Returns:

the related topic unique identification.

final boolean Subscription.isDurable ()

Returns true if the subscription is durable, false otherwise.

Returns:

true if the subscription is durable, false otherwise.

4.18. ClusterConnectionFactory

Class org.objectweb.joram.client.jms.admin.ClusterConnectionFactory

Public Member Functions

```
void addConnectionFactory (ConnectionFactory cf)
void addConnectionFactory (String location, ConnectionFactory cf)
final Connection createConnection () throws JMSException
final Connection createConnection (String name, String password) throws JMSException
Hashtable getCluster ()
```

Detailed Description

A base class for clustered connection factories.

Constructor Documentation

ClusterConnectionFactory.ClusterConnectionFactory ()
Constructs an empty clustered connection factory.

Member Function Documentation

void ClusterConnectionFactory.addConnectionFactory (ConnectionFactory cf)

Adds a connection factory to the cluster. The object will be added with a key equals to the location property. By default, the location value is set to the hostname of corresponding server. Be careful, the object should be rebind after modification.

Parameters:

cf	the ConnectionFactory
----	-----------------------

void ClusterConnectionFactory.addConnectionFactory (String location, ConnectionFactory cf)

Adds a connection factory to the cluster with the specified location key. Be careful, the object should be rebind after modification.

Parameters:

location	the location key
cf	the ConnectionFactory

final Connection ClusterConnectionFactory.createConnection () throws JMSException

Creates a connection with the default user identity. It chooses a ConnectionFactory depending of the location property, then creates the related Connection .
API method, see javax.jms.ConnectionFactory.

Exceptions:

JMSecurityException	If the default identification is incorrect.
IllegalStateException	If the server is not listening.

**final Connection ClusterConnectionFactory.createConnection (String name, String password)
throws JMSException**

Creates a connection with the specified user identity. It chooses a ConnectionFactory depending of the location property, then creates the related Connection .

API method, see javax.jms.ConnectionFactory.

Parameters:

name	the caller's user name
password	the caller's password

Exceptions:

JMSecurityException	If the user identification is incorrect.
IllegalStateException	If the server is not listening.

Hashtable ClusterConnectionFactory.getCluster ()

Returns:

the cluster hashtable.

4.19. ClusterDestination

Class org.objectweb.joram.client.jms.admin.ClusterDestination:

Public Member Functions

```
void addDestination (Destination dest)
void addDestination (String location, Destination dest)
String getName ()
byte getType ()
void setReader (User user) throws ConnectException, AdminException
void setWriter (User user) throws ConnectException, AdminException
void setFreeReading () throws ConnectException, AdminException
void setFreeWriting () throws ConnectException, AdminException
void toReference (Reference ref) throws NamingException
void fromReference (Reference ref) throws NamingException
```

Detailed Description

A base class for clustered destinations.

Constructor Documentation

ClusterDestination.ClusterDestination ()

Constructs an empty clustered destination.

ClusterDestination.ClusterDestination (Hashtable cluster)

Constructs a cluster destination.

Parameters:

cluster	Hashtable of the cluster agent destination.
---------	---

Member Function Documentation

void ClusterDestination.addDestination (Destination dest)

Adds a destination to the cluster. The object will be added with a key equals to the location property. Be careful, the object should be rebind after modification.

Parameters:

dest	the Destination
------	-----------------

void ClusterDestination.addDestination (String location, Destination dest)

Adds a destination to the cluster with the specified location key. By default, the location value is set to the String "server::i", where i is the server id. of the destination. Be careful, the object should be rebind after modification.

Parameters:

location	the location key
dest	the Destination

String ClusterDestination.getName ()

Returns the name of the destination.

byte ClusterDestination.getType ()

Returns the type of the destination: queue or topic, temporary or not.

void ClusterDestination.setFreeReading () throws ConnectException, AdminException

Administration method setting free reading access to this destination.

The request fails if this destination is deleted server side.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

void ClusterDestination.setFreeWriting () throws ConnectException, AdminException

Administration method setting free writing access to this destination.

The request fails if this destination is deleted server side.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

void ClusterDestination.setReader (User user) throws ConnectException, AdminException

Administration method setting a given user as a reader on this destination.

The request fails if this destination is deleted server side.

Parameters:

user	User to be set as a reader.
------	-----------------------------

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

void ClusterDestination.setWriter (User user) throws ConnectException, AdminException

Administration method setting a given user as a writer on this destination.

The request fails if this destination is deleted server side.

Parameters:

user	User to be set as a writer.
------	-----------------------------

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

4.20. ClusterQueue

Class org.objectweb.joram.client.jms.admin.ClusterQueue

Public Member Functions

String getQueueName () throws JMSEException

Detailed Description

Clustered queue.

Constructor Documentation

ClusterQueue.ClusterQueue ()

Constructs an empty cluster queue.

ClusterQueue.ClusterQueue (Hashtable cluster)

Constructs a cluster queue.

Parameters:

cluster	Hashtable of the cluster agent destination.
---------	---

Member Function Documentation

String ClusterQueue.getQueueName () throws JMSEException

API method.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

4.21. ClusterTopic

Class org.objectweb.joram.client.jms.admin.ClusterTopic:

Public Member Functions

String getTopicName () throws JMSEException

Detailed Description

Clustered topic.

Constructor Documentation

ClusterTopic.ClusterTopic ()

Constructs an empty cluster topic.

ClusterTopic.ClusterTopic (Hashtable cluster)

Constructs a cluster topic.

Parameters:

cluster	Hashtable of the cluster agent destination.
---------	---

Member Function Documentation

String ClusterTopic.getTopicName () throws JMSException

API method.

Exceptions:

JMSException	Actually never thrown.
--------------	------------------------

4.22. DeadMQueue

Class org.objectweb.joram.client.jms.admin.DeadMQueue

Static Public Member Functions

static Queue create () throws ConnectException, AdminException

static Queue create (int serverId) throws ConnectException, AdminException

static Queue create (int serverId, String name) throws ConnectException, AdminException

Detailed Description

The DeadMQueue class allows administrators to manipulate dead message queues.

Deprecated:

Since Joram 5.2.2 the DeadMQueue is a simple Queue.

Member Function Documentation

static Queue DeadMQueue.create () throws ConnectException, AdminException [static]

Admin method creating and deploying a dead message queue on the local server.

The request fails if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

static Queue admin.DeadMQueue.create (int serverId) throws ConnectException, AdminException [static]

Admin method creating and deploying a dead message queue on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
----------	---

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

static Queue DeadMQueue.create (int serverId, String name) throws ConnectException, AdminException [static]

Admin method creating and deploying a dead message queue on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

4.23. XALocalConnectionFactory

Class org.objectweb.joram.client.jms.local.XALocalConnectionFactory
Extends org.objectweb.joram.client.jms.XAConnectionFactory

Static Public Member Functions

static javax.jms.XAConnectionFactory create ()

Detailed Description

An XALocalConnectionFactory instance is a factory of local connections dedicated to XA communication.

Deprecated:

Replaced next to Joram 5.2.1 by LocalConnectionFactory.

Member Function Documentation

static javax.jms.XAConnectionFactory XALocalConnectionFactory.create () [static]

Administration method creating a javax.jms.XAConnectionFactory instance for creating local connections.

4.24. XATcpConnectionFactory

Class org.objectweb.joram.client.jms.tcp.XATcpConnectionFactory
 Extends org.objectweb.joram.client.jms.XAConnectionFactory

Static Public Member Functions

```
static javax.jms.XAConnectionFactory create () throws java.net.ConnectException
static javax.jms.XAConnectionFactory create (String host, int port)
static javax.jms.XAConnectionFactory create (String host, int port, String reliableClass)
```

Detailed Description

An XATcpConnectionFactory instance is a factory of TCP connections dedicated to XA communication.

Deprecated:

Replaced next to Joram 5.2.1 by TcpConnectionFactory.

Member Function Documentation

static javax.jms.XAConnectionFactory XATcpConnectionFactory.create () throws java.net.ConnectException [static]

Administration method creating a javax.jms.XAConnectionFactory instance for creating TCP connections with the default server.

Exceptions:

java.net.ConnectException	If the admin connection is closed or broken.
---------------------------	--

See also:

getDefaultServerHost()
 getDefaultServerPort()

static javax.jms.XAConnectionFactory XATcpConnectionFactory.create (String host, int port) [static]

Administration method creating a javax.jms.XAConnectionFactory instance for creating TCP connections with a given server.

Parameters:

host	Name or IP address of the server's host.
port	Server's listening port.

static javax.jms.XAConnectionFactory XATcpConnectionFactory.create (String host, int port, String reliableClass) [static]

Admin method creating a javax.jms.XAConnectionFactory instance for creating TCP connections with a given server.

Parameters:

host	Name or IP address of the server's host.
port	Server's listening port.
reliableClass	Reliable class name.

4.25. XAConnection

Class org.objectweb.joram.client.jms.XAConnection

Extends Connection
Implements javax.jms.XAConnection

Public Member Functions

`javax.jms.Session createSession (boolean transacted, int acknowledgeMode) throws JMSEException`
`javax.jms.XASession createXASession () throws JMSEException`

Detailed Description

Connection used within global transactions; an instance of this class acts as a resource manager. The XAConnection class extends the capability of Connection by providing an XASession. This class offers support to transactional environments. Client programs are strongly encouraged to use the transactional support available in their environment, rather than use these XA interfaces directly.

Member Function Documentation

javax.jms.Session XAConnection.createSession (boolean transacted, int acknowledgeMode) throws JMSEException

API method. Creates a non-XA session.

Parameters:

transacted	indicates whether the session is transacted.
acknowledgeMode	indicates whether the consumer or the client will acknowledge any messages it receives; ignored if the session is transacted. Legal values are Session.AUTO_ACKNOWLEDGE, Session.CLIENT_ACKNOWLEDGE, and Session.DUPS_OK_ACKNOWLEDGE.

Returns:

a newly created Session.

Exceptions:

IllegalStateException	If the connection is closed.
JMSEException	In case of an invalid acknowledge mode.

javax.jms.XASession XAConnection.createXASession () throws JMSEException

API method. Creates an XASession object.

Returns:

a newly created XASession.

Exceptions:

IllegalStateException	If the connection is closed.
-----------------------	------------------------------

4.26. XASession

Class org.objectweb.joram.client.jms.XASession
Implements javax.jms.XASession

Public Member Functions

`javax.jms.Session getSession () throws JMSEException`
`javax.transaction.xa.XAResource getXAResource ()`
`boolean getTransacted () throws JMSEException`
`void commit () throws JMSEException`
`void rollback () throws JMSEException`
`void recover () throws JMSEException`

void close () throws JMSException

Detailed Description

Implements the javax.jms.XASession interface.

An XA session actually extends the behaviour of a normal session by providing an XA resource representing it to a Transaction Manager, so that it is part of a distributed transaction. The XASession wraps what looks like a "normal" Session object. This object takes care of producing and consuming messages, the actual sendings and acknowledgement being managed by this XA wrapper.

This class offers support to transactional environments. Client programs are strongly encouraged to use the transactional support available in their environment, rather than use these XA interfaces directly.

Member Function Documentation

Most of the methods of the interface are handled directly by the wrapped session, only methods with different behavior are described here.

void XASession.close () throws JMSException

API method. Closes the session.

See also:

Session::close()

Exceptions:

JMSException	Actually never thrown.
--------------	------------------------

void XASession.commit () throws JMSException

API method inherited from session, but intercepted here for forbidding its use in the XA context (as defined by the API).

Exceptions:

IllegalStateException	Systematically thrown.
-----------------------	------------------------

javax.jms.Session XASession.getSession () throws JMSException

API method. Gets the session associated with this XASession.

Returns:

the session object.

Exceptions:

IllegalStateException	If the session is closed.
-----------------------	---------------------------

boolean XASession.getTransacted () throws JMSException

API method. Indicates whether the session is in transacted mode.

Returns:

true.

Exceptions:

IllegalStateException	If the session is closed.
-----------------------	---------------------------

javax.transaction.xa.XAResource XASession.getXAResource ()

API method. Returns an XA resource to the caller.

Returns:

an XA resource.

void XASession.recover () throws JMSException

API method inherited from session, but intercepted here for forbidding its use in the XA context (as defined by the API).

Exceptions:

IllegalStateException	Systematically thrown.
-----------------------	------------------------

void XASession.rollback () throws JMSException

API method inherited from session, but intercepted here for forbidding its use in the XA context (as defined by the API).

Exceptions:

IllegalStateException	Systematically thrown.
-----------------------	------------------------

4.27. PooledConnectionFactory

Class org.objectweb.joram.client.jms.pool.PooledConnectionFactory
Implements ConnectionFactory.

Public Member Functions

```
ConnectionFactory getConnectionFactory ()
int getMaxFreeConnections ()
void setMaxFreeConnections (int maxFreeConnections)
PooledConnectionFactory (ConnectionFactory cf)
PooledConnectionFactory (ConnectionFactory cf, int maxFreeConnections)
Connection createConnection () throws JMSException
synchronized Connection createConnection (String name, String password) throws JMSException
```

Detailed Description

A ConnectionFactory which pools Connection for reuse.

Constructor Documentation

PooledConnectionFactory.PooledConnectionFactory (ConnectionFactory cf)

Creates a new pool for the specified ConnectionFactory.

Parameters:

cf	The ConnectionFactory used to really create the connections.
----	--

PooledConnectionFactory.PooledConnectionFactory (ConnectionFactory cf, int maxFreeConnections)
Creates a new pool for the specified ConnectionFactory.

Parameters:

cf	The ConnectionFactory used to really create the connections.
maxFreeConnections	The maximum number of free connections for an identity in the pool.

Member Function Documentation

Connection PooledConnectionFactory.createConnection () throws JMSException

API method, creates a connection with the default user identity. The connection is created in stopped mode.

Returns:

a newly created connection.

See also:

`javax.jms.ConnectionFactory::createConnection()`

Exceptions:

JMSecurityException	If the default identification is incorrect.
IllegalStateException	If the server is not listening.

synchronized Connection PooledConnectionFactory.createConnection (String name, String password) throws JMSException

API method, creates a connection with the specified user identity. The connection is created in stopped mode.

Parameters:

name	the caller's user name.
password	the caller's password.

Returns:

a newly created connection.

See also:

`javax.jms.ConnectionFactory::createConnection(String, String)`

Exceptions:

JMSecurityException	If the user identification is incorrect.
IllegalStateException	If the server is not listening.

ConnectionFactory PooledConnectionFactory.getConnection ()

Returns the underlying ConnectionFactory used to create the connections. This ConnectionFactory object allows to configure the created connections.

Returns:

the underlying ConnectionFactory.

int PooledConnectionFactory.getMaxFreeConnections ()

Returns the maximum number of free connections for an identity in the pool.

Returns:

The maximum number of free connections in the pool.

void PooledConnectionFactory.setMaxFreeConnections (int maxFreeConnections)

Sets the maximum number of free connections for an identity in the pool.

Parameters:

maxFreeConnections	the maximum number of free connections to set
--------------------	---

4.28. PooledConnection

Class org.objectweb.joram.client.jms.pool.PooledConnection
Implements Connection.

Public Member Functions

PooledConnection (PooledConnectionFactory pcf, String name, String password) throws JMSException
void close () throws JMSException

```

ConnectionConsumer createConnectionConsumer (Destination dest, String selector, ServerSessionPool
sessionPool, int maxMessages) throws JMSEException
ConnectionConsumer createDurableConnectionConsumer (Topic topic, String subName, String selector,
ServerSessionPool sessPool, int maxMessages) throws JMSEException
Session createSession (boolean transacted, int acknowledgeMode) throws JMSEException
String getClientID () throws JMSEException
ExceptionListener getExceptionListener () throws JMSEException
ConnectionMetaData getMetaData () throws JMSEException
void setClientID (String clientID) throws JMSEException
void setExceptionListener (ExceptionListener listener) throws JMSEException
void start () throws JMSEException
void stop () throws JMSEException

```

Detailed Description

Implements a pooled connection.

Constructor Documentation

PooledConnection.PooledConnection (PooledConnectionFactory pcf, String name, String password) throws JMSEException

Creates a new pooled connection.

Parameters:

pcf	The pooled ConnectionFactory.
name	The name of the authentified user for this connection.
password	The password of the authentified user for this connection.

Exceptions:

JMSEException	An error occurs during the connection.
---------------	--

Member Function Documentation

void PooledConnection.close () throws JMSEException

Close the pooled connection, depending of the pool state this can results in the real closing of the connection of the inserting in the pool of idle connections.

See also:

javax.jms.Connection::close()

ConnectionConsumer PooledConnection.createConnectionConsumer (Destination dest, String selector, ServerSessionPool sessionPool, int maxMessages) throws JMSEException

See also:

javax.jms.Connection::createConnectionConsumer(javax.jms.Destination, java.lang.String, javax.jms.ServerSessionPool, int)

ConnectionConsumer PooledConnection.createDurableConnectionConsumer (Topic topic, String subName, String selector, ServerSessionPool sessPool, int maxMessages) throws JMSEException

See also:

javax.jms.Connection::createDurableConnectionConsumer(javax.jms.Topic, java.lang.String, java.lang.String, javax.jms.ServerSessionPool, int)

Session PooledConnection.createSession (boolean transacted, int acknowledgeMode) throws JMSEException

See also:
javax.jms.Connection::createSession(boolean, int)

String PooledConnection.getClientID () throws JMSEException

See also:
javax.jms.Connection::getClientID()

ExceptionListener PooledConnection.getExceptionListener () throws JMSEException

See also:
javax.jms.Connection::getExceptionListener()

ConnectionMetaData PooledConnection.getMetaData () throws JMSEException

See also:
javax.jms.Connection::getMetaData()

void PooledConnection.setClientID (String clientID) throws JMSEException

See also:
javax.jms.Connection::setClientID(java.lang.String)

void PooledConnection.setExceptionListener (ExceptionListener listener) throws JMSEException

See also:
javax.jms.Connection::setExceptionListener(javax.jms.ExceptionListener)

void PooledConnection.start () throws JMSEException

See also:
javax.jms.Connection::start()

void PooledConnection.stop () throws JMSEException

See also:
javax.jms.Connection::stop()

4.29. ConnectionPool

Class org.objectweb.joram.client.jms.pool.ConnectionPool

Public Member Functions

int getMaxFreeConnections ()
void setMaxFreeConnections (int maxFreeConnections)

Detailed Description

Implements a pool of connection resulting of a unique ConnectionFactory object. Connections are sorted by identity.

Constructor Documentation

ConnectionPool.ConnectionPool (int maxFreeConnections)

Creates a new pool of connections.

Parameters:

maxFreeConnectio ns	the maximum number of free connections for an identity in the pool.
------------------------	---

Member Function Documentation

int ConnectionPool.getMaxFreeConnections ()

Gets the maximum number of free connections for an identity in the pool.

Returns:

the maxFreeConnections

void ConnectionPool.setMaxFreeConnections (int maxFreeConnections)

Returns the maximum number of free connections for an identity in the pool.

Parameters:

maxFreeConnectio ns	the maxFreeConnections to set
------------------------	-------------------------------

5. Messages API

Class Reference

This chapter describes the implementation of the JMS 1.1 Message interfaces.

5.1. Message

Class org.objectweb.joram.client.jms.Message
Implements javax.jms.Message

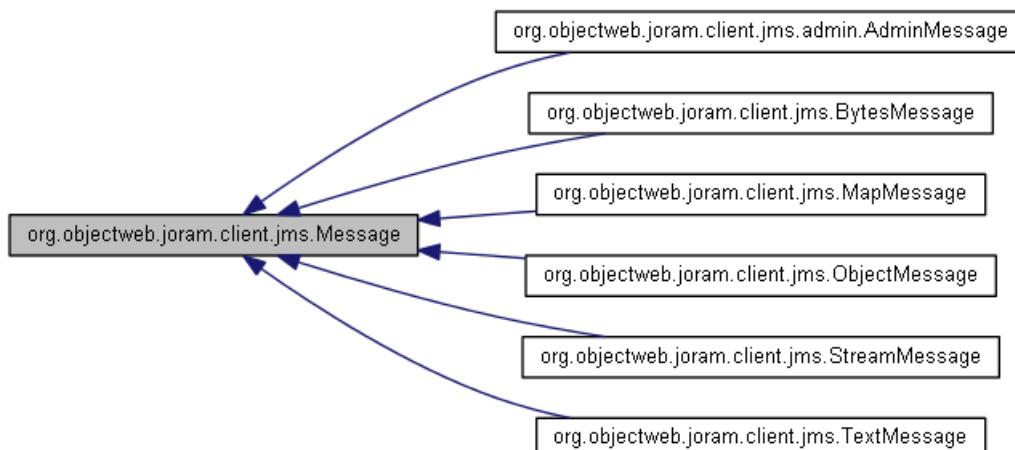


Figure 5.1 - Inheritance diagram for Message class

Public Member Functions

```

void acknowledge () throws JMSEException
void clearBody () throws JMSEException
final String getJMSMessageID () throws JMSEException
final void setJMSMessageID (String id) throws JMSEException
final int getJMSPriority () throws JMSEException
final void setJMSPriority (int priority) throws JMSEException
final javax.jms.Destination getJMSDestination () throws JMSEException
final void setJMSDestination (javax.jms.Destination dest) throws JMSEException
final long getJMSExpiration () throws JMSEException
final void setJMSExpiration (long expiration) throws JMSEException
final boolean getJMSRedelivered () throws JMSEException
final void setJMSRedelivered (boolean redelivered) throws JMSEException
final javax.jms.Destination getJMSReplyTo () throws JMSEException
final void setJMSReplyTo (javax.jms.Destination replyTo) throws JMSEException
final long getJMSTimestamp () throws JMSEException
final void setJMSTimestamp (long timestamp) throws JMSEException
  
```

```

final String getJMSCorrelationID () throws JMSEException
final void setJMSCorrelationID (String correlationID) throws JMSEException
final byte[] getJMSCorrelationIDAsBytes () throws JMSEException
final void setJMSCorrelationIDAsBytes (byte[] correlationID)
final int getJMSDeliveryMode () throws JMSEException
final void setJMSDeliveryMode (int deliveryMode) throws JMSEException
final String getJMSType () throws JMSEException
final void setJMSType (String type) throws JMSEException
final void clearProperties () throws JMSEException
final void resetPropertiesRO () throws JMSEException
final boolean propertyExists (String name) throws JMSEException
void getProperties (Map h)
final Enumeration getPropertyNames () throws JMSEException
final void setBooleanProperty (String name, boolean value) throws JMSEException
final void setByteProperty (String name, byte value) throws JMSEException
final void setShortProperty (String name, short value) throws JMSEException
final void setIntProperty (String name, int value) throws JMSEException
final void setLongProperty (String name, long value) throws JMSEException
final void setFloatProperty (String name, float value) throws JMSEException
final void setDoubleProperty (String name, double value) throws JMSEException
final void setStringProperty (String name, String value) throws JMSEException
final void setObjectProperty (String name, Object value) throws JMSEException
final boolean getBooleanProperty (String name) throws JMSEException
final byte getByteProperty (String name) throws JMSEException
final short getShortProperty (String name) throws JMSEException
final int getIntProperty (String name) throws JMSEException
final long getLongProperty (String name) throws JMSEException
final float getFloatProperty (String name) throws JMSEException
final double getDoubleProperty (String name) throws JMSEException
final String getStringProperty (String name) throws JMSEException
final Object getObjectProperty (String name) throws JMSEException
final int getCompressedMinSize()
final void setCompressedMinSize(int compressedMinSize)
final int getCompressionLevel()
final void setCompressionLevel(int compressionLevel)

```

Static Public Member Functions

static Message convertJMSMessage (javax.jms.Message jmsMsg) throws JMSEException

Detailed Description

Implements the javax.jms.Message interface.

A Joram message encapsulates a proprietary message which is used for effective MOM transport facility. It defines the message header and properties, and the acknowledge method used for all messages.

JMS messages are composed of the following parts:

- Header - All messages support the same set of header fields. Header fields contain values used by both clients and providers to identify and route messages.
- Properties - Each message contains a built-in facility for supporting application-defined property values. Properties provide an efficient mechanism for supporting message filtering.
- Body - The JMS API defines several types of message body, which cover the majority of messaging styles currently in use.

The JMS API defines five types of message body:

- Stream - A StreamMessage object's message body contains a stream of primitive values.
- Map - A MapMessage object's message body contains a set of name-value pairs, where names are String objects, and values are Java primitives.
- Text - A TextMessage object's message body contains a java.lang.String object.
- Object - An ObjectMessage object's message body contains a Serializable Java object.
- Bytes - A BytesMessage object's message body contains a stream of uninterpreted bytes.

Member Function Documentation

void Message.acknowledge () throws JMSEException

API method. Acknowledges all previously consumed messages of the session of this consumed message. All consumed JMS messages support the acknowledge method for use when a client has specified that its JMS session's consumed messages are to be explicitly acknowledged. By invoking acknowledge on a consumed message, a client implicitly acknowledges all messages consumed by the session that the message was delivered to.

Calls to acknowledge are ignored for both transacted sessions and sessions specified to use implicit acknowledgement modes.

Messages that have been received but not acknowledged may be redelivered.

Exceptions:

IllegalStateException	If the session is closed.
JMSEException	If the acknowledgement fails for any other reason.

void Message.clearBody () throws JMSEException

API method. Clears the message's body.

Calling this method leaves the message body in the same state as an empty body in a newly created message.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

final void Message.clearProperties () throws JMSEException

API method. Clears the message's properties.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

static Message Message.convertJMSMessage (javax.jms.Message jmsMsg) throws JMSEException [static]

Converts a non-Joram JMS message into a Joram message.

Parameters:

jmsMsg	A JMS message.
--------	----------------

Returns:

a Joram message.

Exceptions:

JMSEException	If an error occurs while building the message.
---------------	--

final boolean Message.getBooleanProperty (String name) throws JMSEException

API method. Returns the value of the boolean property with the specified name.

Parameters:

name	The property name.
------	--------------------

Returns:

the property value for the specified name.

Exceptions:

MessageFormatException	If the property type is invalid.
JMSEException	If the name is invalid.

final byte Message.getByteProperty (String name) throws JMSEException

API method. Returns the value of the byte property with the specified name.

Parameters:

name	The property name.
------	--------------------

Returns:

the property value for the specified name.

Exceptions:

MessageFormatException	If the property type is invalid.
JMSEException	If the name is invalid.

final double Message.getDoubleProperty (String name) throws JMSEException

API method. Returns the value of the double property with the specified name.

Parameters:

name	The property name.
------	--------------------

Returns:

the property value for the specified name.

Exceptions:

MessageFormatException	If the property type is invalid.
JMSEException	If the name is invalid.

final float Message.getFloatProperty (String name) throws JMSEException

API method. Returns the value of the float property with the specified name.

Parameters:

name	The property name.
------	--------------------

Returns:

the property value for the specified name.

Exceptions:

MessageFormatException	If the property type is invalid.
JMSEException	If the name is invalid.

final int Message.getIntProperty (String name) throws JMSEException

API method. Returns the value of the int property with the specified name.

Parameters:

name	The property name.
------	--------------------

Returns:

the property value for the specified name.

Exceptions:

MessageFormatException	If the property type is invalid.
JMSEException	If the name is invalid.

final String Message.getJMSCorrelationID () throws JMSEException

API method. Returns the message correlation identifier.

Returns:

the correlation ID for the message.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

final byte [] Message.getJMSCorrelationIDAsBytes () throws JMSEException

API method. Gets the correlation ID as an array of bytes for the message.

Returns:

the correlation ID for the message as an array of bytes.

Exceptions:

MessageFormatException	In case of a problem while retrieving the field.
------------------------	--

final int Message.getJMSSDeliveryMode () throws JMSEException

API method. Gets the DeliveryMode value specified for this message.

The delivery modes supported are DeliveryMode.PERSISTENT and DeliveryMode.NON_PERSISTENT.

Returns:

the delivery mode for this message.

See also:

javax.jms.DeliveryMode

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

final javax.jms.Destination Message.getJMSSDestination () throws JMSEException

API method. Returns the message destination. This field is set by the provider at sending, it contains the destination to which the message is being sent.

When a message is sent, this field is ignored. After completion of the send or publish method, the field holds the destination specified by the method.

When a message is received, its JMSSDestination value must be equivalent to the value assigned when it was sent. This field can be overloaded for received messages.

Returns:

the destination of this message.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

final long Message.getJMSEExpiration () throws JMSEException

API method. Returns the message expiration time.

When a message is sent, the JMSEExpiration header field is ignored. After completion of the send or publish method, it holds the expiration time of the message. This is the sum of the time-to-live value specified by the client and the GMT at the time of the send or publish.

If the time-to-live is specified as zero, JMSEExpiration is set to zero to indicate that the message does not expire.

When a message's expiration time is reached, it is either discarded or forwarded to a DeadMessageQueue.

Returns:

the time the message expires, which is the sum of the time-to-live value specified by the client and the GMT at the time of the send.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

final String Message.getJMSSMessageID () throws JMSEException

API method. Returns the unique message identifier.

The JMSSMessageID header field contains a value that uniquely identifies each message sent by Joram. When a message is sent, JMSSMessageID is ignored, when the send or publish method returns, it contains an unique identifier assigned by Joram.

All JMSSMessageID values starts with the prefix 'ID:'.

Returns:

the unique message identifier.

Exceptions:

JMSException	Actually never thrown.
--------------	------------------------

final int Message.getJMSPriority () throws JMSException

API method. Returns the message priority.

The JMS API defines ten levels of priority value, with 0 as the lowest priority and 9 as the highest. In addition, clients should consider priorities 0-4 as gradations of normal priority and priorities 5-9 as gradations of expedited priority.

Default priority is defined by Message.DEFAULT_PRIORITY.

Returns:

the message priority.

Exceptions:

JMSException	Actually never thrown.
--------------	------------------------

final boolean Message.getJMSRedelivered () throws JMSException

API method. Gets an indication of whether this message is being redelivered.

If a client receives a message with the JMSRedelivered field set, it can access the JMSXDeliveryCount property to determine the number of attempts to deliver this message.

Returns:

true if this message is being redelivered.

Exceptions:

JMSException	Actually never thrown.
--------------	------------------------

final javax.jms.Destination Message.getJMSSelector () throws JMSException

API method. Gets the Destination object to which a reply to this message should be sent.

Returns:

Destination to which to send a response to this message.

Exceptions:

JMSException	Actually never thrown.
--------------	------------------------

final long Message.getJMSTimestamp () throws JMSException

API method. Returns the message time stamp.

The JMSTimestamp header field contains the time a message was handed off to Joram to be sent. It is not the time the message was actually transmitted, because the actual send may occur later due to transactions or other client-side queueing of messages.

When a message is sent, JMSTimestamp is ignored. When the send or publish method returns, it contains a time value somewhere in the interval between the call and the return.

Since timestamps take some effort to create and increase a message's size, some Joram allows to optimize message overhead if they are given a hint that the timestamp is not used by an application. By calling the MessageProducer.setDisableMessageTimestamp method, a JMS client enables this potential optimization for all messages sent by that message producer.

Returns:

the message timestamp.

Exceptions:

JMSException	Actually never thrown.
--------------	------------------------

final String Message.getJMSType () throws JMSException

API method. Gets the message type identifier supplied by the client when the message was sent.

Returns:

the message type

Exceptions:

JMSException	Actually never thrown.
--------------	------------------------

final long Message.getLongProperty (String name) throws JMSException

API method. Returns the value of the long property with the specified name.

Parameters:

name	The property name.
------	--------------------

Returns:

the property value for the specified name.

Exceptions:

MessageFormatException	If the property type is invalid.
JMSException	If the name is invalid.

final Object Message.getObjectProperty (String name) throws JMSException

API method. Returns the value of the object property with the specified name.

This method can be used to return, in objectified format, an object that has been stored as a property in the message with the equivalent setObjectProperty method call, or its equivalent primitive settypeProperty method.

Parameters:

name	The property name.
------	--------------------

Returns:

the Java object property value with the specified name, in objectified format; if there is no property by this name, a null value is returned.

Exceptions:

JMSException	If the name is invalid.
--------------	-------------------------

void Message.getProperties (Map h)

Copies all of the mappings from the properties of this message to the specified map. These mappings will replace any mappings that this Map had for any of the keys currently in the properties.

final Enumeration Message.getPropertyNames () throws JMSException

API method. Returns an Enumeration of all the property names.

Note that JMS standard header fields are not considered properties and are not returned in this enumeration.

Returns:

An enumeration of all the names of property values.

Exceptions:

JMSException	Actually never thrown.
--------------	------------------------

final short Message.getShortProperty (String name) throws JMSException

API method.

Parameters:

name	The property name.
------	--------------------

Returns:

the property value for the specified name.

Exceptions:

MessageFormatException	If the property type is invalid.
JMSEException	If the name is invalid.

final String Message.getStringProperty (String name) throws JMSEException

API method. Returns the value of the String property with the specified name.

Parameters:

name	The property name.
------	--------------------

Returns:

the property value for the specified name.

Exceptions:

MessageFormatException	If the property type is invalid.
JMSEException	If the name is invalid.

final boolean Message.propertyExists (String name) throws JMSEException

API method. Indicates whether a property value exists.

Parameters:

name	the name of the property to test.
------	-----------------------------------

Returns:

true if the property exists.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

final void Message.resetPropertiesRO () throws JMSEException

Resets the read-only flag, in order to allow the modification of message properties.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

final void Message.setBooleanProperty (String name, boolean value) throws JMSEException

API method. Sets a boolean property value with the specified name into the message.

Parameters:

name	The property name.
value	The property value.

Exceptions:

MessageNotWritableException	If the message is read-only.
JMSEException	If the property name is invalid.

final void Message.setProperty (String name, byte value) throws JMSEException

API method. Sets a byte property value with the specified name into the message.

Parameters:

name	The property name.
value	The property value.

Exceptions:

MessageNotWritableException	If the message is read-only.
JMSEException	If the property name is invalid.

final void Message.setDoubleProperty (String name, double value) throws JMSEException

API method. Sets a double property value with the specified name into the message.

Parameters:

name	The property name.
value	The property value.

Exceptions:

MessageNotWritableException	If the message is read-only.
JMSEException	If the property name is invalid.

final void Message.setFloatProperty (String name, float value) throws JMSEException

API method. Sets a float property value with the specified name into the message.

Parameters:

name	The property name.
value	The property value.

Exceptions:

MessageNotWritableException	If the message is read-only.
JMSEException	If the property name is invalid.

final void Message.setIntProperty (String name, int value) throws JMSEException

API method. Sets an int property value with the specified name into the message.

Parameters:

name	The property name.
value	The property value.

Exceptions:

MessageNotWritableException	If the message is read-only.
JMSEException	If the property name is invalid.

final void Message.setJMSCorrelationID (String correlationID) throws JMSEException

API method. Sets the correlation ID for the message.

A client can use the JMSCorrelationID header field to link one message with another. A typical use is to link a response message with its request message.

The use of a byte[] value for JMSCorrelationID is non-portable.

Parameters:

correlationID	the message ID of a message being referred to.
---------------	--

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

final void Message.setJMSCorrelationIDAsBytes (byte[] correlationID)

API method. Sets the correlation ID as an array of bytes for the message.

The use of a byte[] value for JMSCorrelationID is non-portable.

Parameters:

correlationID	the message ID value as an array of bytes.
---------------	--

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

final void Message.setJMSDeliveryMode (int deliveryMode) throws JMSException

API method. Sets the DeliveryMode value for this message.

JMS providers set this field when a message is sent. This method can be used to change the value for a message that has been received.

Parameters:

deliveryMode	the delivery mode for this message.
--------------	-------------------------------------

Exceptions:

JMSException	If the delivery mode is incorrect.
--------------	------------------------------------

final void Message.setJMSDestination (javax.jms.Destination dest) throws JMSException

API method. Set the message destination.

This field is set when message is sent, this method can only be used to change the value for a message that has been received.

Parameters:

dest	the destination for this message.
------	-----------------------------------

Exceptions:

JMSException	If the destination id not a Joram's one.
--------------	--

final void Message.setJMSExpiration (long expiration) throws JMSException

API method. Sets the message's expiration value.

This field is set when a message is sent, this method can only be used to change the value for a message that has been received.

Parameters:

expiration	the message's expiration time.
------------	--------------------------------

Exceptions:

JMSException	Actually never thrown.
--------------	------------------------

final void Message.setJMSMessageID (String id) throws JMSException

API method. Sets the message identifier.

This field is set when a message is sent, this method can only be used to change the value for a message that has been received.

Parameters:

id	the identifier for this message.
----	----------------------------------

Exceptions:

JMSException	Actually never thrown.
--------------	------------------------

final void Message.setJMSPriority (int priority) throws JMSException

API method. Sets the priority level for this message.

This field is set when a message is sent, this method can be used to change the value for a message that has been received.

Parameters:

priority	the priority of this message.
----------	-------------------------------

Exceptions:

JMSException	If the priority value is incorrect.
--------------	-------------------------------------

final void Message.setJMSRedelivered (boolean redelivered) throws JMSException

API method. Specifies whether this message is being redelivered.

This field is set at the time the message is delivered, this method can only be used to change the value for a message that has been received.

Parameters:

redelivered	an indication of whether this message is being redelivered.
-------------	---

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

final void Message.setJMSReplyTo (javax.jms.Destination replyTo) throws JMSEException

API method. Sets the Destination object to which a reply to this message should be sent.

The JMSReplyTo header field contains the destination where a reply to the current message should be sent. The destination may be either a Queue object or a Topic object.

Parameters:

replyTo	Destination to which to send a response to this message.
---------	--

Exceptions:

JMSEException	If the destination id not a Joram's one.
---------------	--

final void Message.setJMSTimestamp (long timestamp) throws JMSEException

API method. Sets the message timestamp.

This field is set when a message is sent, this method can only be used to change the value for a message that has been received.

Parameters:

timestamp	the timestamp for this message.
-----------	---------------------------------

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

final void Message.setJMSType (String type) throws JMSEException

API method. Sets the message type.

Joram does not define a standard message definition repository, this field can be used freely by the JMS applications.

Parameters:

type	the message type.
------	-------------------

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

final void Message.setLongProperty (String name, long value) throws JMSEException

API method. Sets a long property value with the specified name into the message.

Parameters:

name	The property name.
value	The property value.

Exceptions:

MessageNotWritableException	If the message is read-only.
-----------------------------	------------------------------

JMSEException	If the property name is invalid.
---------------	----------------------------------

final void setObjectProperty (String name, Object value) throws JMSEException

API method. Sets an object property value.

Note that this method works only for the objectified primitive object types (Integer, Double, Long ...) and String objects.

Parameters:

name	The property name.
value	The property value.

Exceptions:

MessageROException	If the message properties are read-only.
MessageFormatException	If the value is not a Java primitive object.
IllegalArgumentException	If the key name is illegal (null or empty string).
MessageFormatException	If the property type is invalid.
MessageNotWritableException	If the message is read-only.
JMSEException	If the property name is invalid, or if the object is invalid.

final void Message.setShortProperty (String name, short value) throws JMSEException

API method. Sets a short property value with the specified name into the message.

Parameters:

name	The property name.
value	The property value.

Exceptions:

MessageNotWritableException	If the message is read-only.
JMSEException	If the property name is invalid.

final void Message.setStringProperty (String name, String value) throws JMSEException

API method. Sets a String property value with the specified name into the message.

Parameters:

name	The property name.
value	The property value.

Exceptions:

MessageNotWritableException	If the message is read-only.
JMSEException	If the property name is invalid.

final int Message.getCompressedMinSize()

Returns the minimum size beyond which this message's body is compressed.

This attribute is inherited from Session at message creation, the default value is 0 (no compression).

Returns:

the minimum size beyond which this message's body is compressed.

See also:

FactoryParameters::compressedMinSize

final void Message.setCompressedMinSize(int compressedMinSize)

Sets the minimum size beyond which this message's body is compressed.

Parameters:

compressedMinSize	the minimum size beyond which this message's body is compressed.
-------------------	--

See also:

FactoryParameters::compressedMinSize

final int Message.getCompressionLevel()

Returns the compression level (0..9) used when this message body is compressed.

This attribute is inherited from Session at message creation, the default value is 1 (Deflater.BEST_SPEED).

Returns:

The compression level.

See also:

FactoryParameters::compressionLevel

final void Message.setCompressionLevel(int compressionLevel)

Sets the compression level (0..9) used when this message body is compressed.

This attribute is inherited from Session at message creation, the default value is 1 (Deflater.BEST_SPEED).

Parameters:

compressedLevel	the compression level (0..9) used when a message body is compressed.
-----------------	--

See also:

FactoryParameters::compressionLevel

5.2. BytesMessage

Class org.objectweb.joram.client.jms.BytesMessage
 extends org.objectweb.joram.client.jms.Message
 implements javax.jms.BytesMessage

Public Member Functions

```
long getBodyLength () throws JMSException
void clearBody () throws JMSException
void writeBoolean (boolean value) throws JMSException
void writeByte (byte value) throws JMSException
void writeBytes (byte[] value) throws JMSException
void writeBytes (byte[] value, int offset, int length) throws JMSException
void writeChar (char value) throws JMSException
void writeDouble (double value) throws JMSException
void writeFloat (float value) throws JMSException
void writeInt (int value) throws JMSException
void writeLong (long value) throws JMSException
void writeShort (short value) throws JMSException
void writeUTF (String value) throws JMSException
void writeObject (Object value) throws JMSException
boolean readBoolean () throws JMSException
byte readByte () throws JMSException
int readUnsignedByte () throws JMSException
short readShort () throws JMSException
int readUnsignedShort () throws JMSException
char readChar () throws JMSException
int readInt () throws JMSException
long readLong () throws JMSException
float readFloat () throws JMSException
double readDouble () throws JMSException
int readBytes (byte[] value) throws JMSException
int readBytes (byte[] value, int length) throws JMSException
String readUTF () throws JMSException
void reset () throws JMSException
```

Detailed Description

Implements the javax.jms.BytesMessage interface.

A BytesMessage object is used to send a message containing a stream of uninterpreted bytes. It inherits from the Message interface and adds a bytes message body. The BytesMessage methods are based largely on those found in java.io.DataInputStream and java.io.DataOutputStream.

The primitive types can be written explicitly using methods for each type. They may also be written generically as objects. For instance, a call to BytesMessage.writeInt(6) is equivalent to BytesMessage.writeObject(new Integer(6)).

When the message is first created, and when clearBody is called, the body of the message is in write-only mode. After the first call to reset has been made, the message body is in read-only mode. After a message has been sent, the client that sent it can retain and modify it without affecting the message that has been sent. The same message object can be sent multiple times. When a message has been received, the provider has called reset so that the message body is in read-only mode for the client.

If clearBody is called on a message in read-only mode, the message body is cleared and the message is in write-only mode.

If a client attempts to read a message in write-only mode, a MessageNotReadableException is thrown.

If a client attempts to write a message in read-only mode, a MessageNotWriteableException is thrown.

Member Function Documentation

void BytesMessage.clearBody () throws JMSException

API method. Clears out the message body.

Calling this method leaves the message body in the same state as an empty body in a newly created message.

Exceptions:

JMSException	In case of an error while closing the output or input streams.
--------------	--

long BytesMessage.getBodyLength () throws JMSException

API method. Gets the number of bytes of the message body when the message is in read-only mode. The value returned can be used to allocate a byte array. The value returned is the entire length of the message body, regardless of where the pointer for reading the message is currently located.

Returns:

the number of bytes in the message's body.

Exceptions:

MessageNotReadableException	If the message is WRITE-ONLY.
-----------------------------	-------------------------------

boolean BytesMessage.readBoolean () throws JMSException

API method. Reads a boolean from the bytes message stream.

Returns:

the value read

Exceptions:

MessageNotReadableException	If the message body is write-only.
JMSException	If an exception occurs while reading the bytes.

byte BytesMessage.readByte () throws JMSException

API method. Reads a byte from the bytes message stream.

Returns:

the value read

Exceptions:

MessageNotReadableException	If the message body is write-only.
JMSEException	If an exception occurs while reading the bytes.

int BytesMessage.readBytes (byte[] value) throws JMSEException

API method. Reads up to value.length bytes from the bytes message stream. A subsequent call reads the next increment, and so on.

A return value of the total number of bytes read less than the length of the array indicates that there are no more bytes left to be read from the stream. The next read of the stream returns -1.

Parameters:

value	the buffer into which the data is read.
-------	---

Returns:

the total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached.

Exceptions:

MessageNotReadableException	If the message body is write-only.
JMSEException	If an exception occurs while reading the bytes.

int BytesMessage.readBytes (byte[] value, int length) throws JMSEException

API method. Reads up to length bytes of the bytes message stream. A subsequent call reads the next increment, and so on.

A return value of the total number of bytes read less than the length parameter indicates that there are no more bytes left to be read from the stream. The next read of the stream returns -1.

If length is negative, or length is greater than the length of the array value, then an IndexOutOfBoundsException is thrown. No bytes will be read from the stream for this exception case.

Parameters:

value	the buffer into which the data is read.
length	the number of bytes to read; must be less than or equal to value.length.

Returns:

the total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached.

Exceptions:

MessageNotReadableException	If the message body is write-only.
JMSEException	If an exception occurs while reading the bytes.

char BytesMessage.readChar () throws JMSEException

API method. Reads a char from the bytes message stream.

Returns:

the value read

Exceptions:

MessageNotReadableException	If the message body is write-only.
JMSEException	If an exception occurs while reading the bytes.

double BytesMessage.readDouble () throws JMSEException

API method. Reads a double from the bytes message stream.

Returns:

the value read

Exceptions:

MessageNotReadableException	If the message body is write-only.
JMSEException	If an exception occurs while reading the bytes.

float BytesMessage.readFloat () throws JMSEException

API method. Reads a float from the bytes message stream.

Returns:

the value read

Exceptions:

MessageNotReadableException	If the message body is write-only.
JMSEException	If an exception occurs while reading the bytes.

int BytesMessage.readInt () throws JMSEException

API method. Reads an int from the bytes message stream.

Returns:

the value read

Exceptions:

MessageNotReadableException	If the message body is write-only.
JMSEException	If an exception occurs while reading the bytes.

long BytesMessage.readLong () throws JMSEException

API method. Reads a long from the bytes message stream.

Returns:

the value read

Exceptions:

MessageNotReadableException	If the message body is write-only.
JMSEException	If an exception occurs while reading the bytes.

short BytesMessage.readShort () throws JMSEException

API method. Reads a short from the bytes message stream.

Returns:

the value read

Exceptions:

MessageNotReadableException	If the message body is write-only.
JMSEException	If an exception occurs while reading the bytes.

int BytesMessage.readUnsignedByte () throws JMSEException

API method. Reads an unsigned byte from the bytes message stream.

Returns:

the next byte from the bytes message stream, interpreted as an unsigned 8-bit number.

Exceptions:

MessageNotReadableException	If the message body is write-only.
JMSEException	If an exception occurs while reading the bytes.

int BytesMessage.readUnsignedShort () throws JMSEException

API method. Reads an unsigned short from the bytes message stream.

Returns:

the next two bytes from the bytes message stream, interpreted as an unsigned 16-bit integer.

Exceptions:

MessageNotReadableException	If the message body is write-only.
JMSEException	If an exception occurs while reading the bytes.

String BytesMessage.readUTF () throws JMSEException

API method. Reads a string that has been encoded using a modified UTF-8 format from the bytes message stream.

Returns:

a Unicode string from the bytes message stream.

Exceptions:

MessageNotReadableException	If the message body is write-only.
JMSEException	If an exception occurs while reading the bytes.

void BytesMessage.reset () throws JMSEException

API method. Puts the message body in read-only mode and repositions the stream of bytes to the beginning.

Exceptions:

JMSEException	If an error occurs while closing the output stream.
---------------	---

void BytesMessage.writeBoolean (boolean value) throws JMSEException

API method. Writes a boolean to the bytes message stream as a 1-byte value. The value true is written as the value (byte)1; the value false is written as the value (byte)0.

Parameters:

value	the value to be written.
-------	--------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSEException	If the value could not be written on the stream.

void BytesMessage.writeByte (byte value) throws JMSEException

API method. Writes a byte to the bytes message stream.

Parameters:

value	the value to be written.
-------	--------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSEException	If the value could not be written on the stream.

void BytesMessage.writeBytes (byte[] value) throws JMSEException

API method. Writes a byte array to the bytes message stream.

Parameters:

value	the byte array to be written.
-------	-------------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSEException	If the value could not be written on the stream.

void BytesMessage.writeBytes (byte[] value, int offset, int length) throws JMSEException

API method. Writes a portion of a byte array to the bytes message stream.

Parameters:

value	the byte array to be written.
offset	the initial offset within the byte array
length	the number of bytes to use

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSEException	If the value could not be written on the stream.

void BytesMessage.writeChar (char value) throws JMSEException

API method. Writes a char to the bytes message stream.

Parameters:

value	the value to be written.
-------	--------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSEException	If the value could not be written on the stream.

void BytesMessage.writeDouble (double value) throws JMSEException

API method. Writes a double to the bytes message stream.

Parameters:

value	the value to be written.
-------	--------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSEException	If the value could not be written on the stream.

void BytesMessage.writeFloat (float value) throws JMSEException

API method. Writes a float to the bytes message stream.

Parameters:

value	the value to be written.
-------	--------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSEException	If the value could not be written on the stream.

void BytesMessage.writeInt (int value) throws JMSException

API method. Writes an int to the bytes message stream.

Parameters:

value	the value to be written.
-------	--------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSException	If the value could not be written on the stream.

void BytesMessage.writeLong (long value) throws JMSException

API method. Writes a long to the bytes message stream.

Parameters:

value	the value to be written.
-------	--------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSException	If the value could not be written on the stream.

void BytesMessage.writeObject (Object value) throws JMSException

API method. Writes an object to the bytes message stream.

This method works only for the objectified primitive object types (Integer, Double, Long ...), String objects, and byte arrays.

Parameters:

value	the primitive Java object to be written; it must not be null.
-------	---

Exceptions:

MessageNotWritableException	If the message body is read-only.
MessageFormatException	If the value type is invalid.
JMSException	If the value could not be written on the stream.

void BytesMessage.writeShort (short value) throws JMSException

API method. Writes a short to the bytes message stream.

Parameters:

value	the value to be written.
-------	--------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSException	If the value could not be written on the stream.

void .BytesMessage.writeUTF (String value) throws JMSException

API method. Writes a string to the bytes message stream using UTF-8 encoding in a machine-independent manner.

Parameters:

value	the String value to be written.
-------	---------------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSException	If the value could not be written on the stream.

5.3. MapMessage

Class org.objectweb.joram.client.jms.MapMessage
 extends org.objectweb.joram.client.jms.Message
 implements javax.jms.MapMessage

Public Member Functions

```
void clearBody () throws JMSEException
void setBoolean (String name, boolean value) throws JMSEException
void setByte (String name, byte value) throws JMSEException
void setBytes (String name, byte[] value) throws JMSEException
void setBytes (String name, byte[] value, int offset, int length) throws JMSEException
void setChar (String name, char value) throws JMSEException
void setDouble (String name, double value) throws JMSEException
void setFloat (String name, float value) throws JMSEException
void setInt (String name, int value) throws JMSEException
void setLong (String name, long value) throws JMSEException
void setShort (String name, short value) throws JMSEException
void setString (String name, String value) throws JMSEException
void setObject (String name, Object value) throws JMSEException
boolean getBoolean (String name) throws JMSEException
byte getByte (String name) throws JMSEException
byte[] getBytes (String name) throws JMSEException
char getChar (String name) throws JMSEException
double getDouble (String name) throws JMSEException
float getFloat (String name) throws JMSEException
int getInt (String name) throws JMSEException
long getLong (String name) throws JMSEException
Object getObject (String name) throws JMSEException
short getShort (String name) throws JMSEException
String getString (String name) throws JMSEException
boolean itemExists (String name) throws JMSEException
Enumeration getMapNames () throws JMSEException
```

Detailed Description

Implements the javax.jms.MapMessage interface.

A MapMessage object is used to send a set of name-value pairs. The names are String objects, and the values are primitive Java data types. The names must have a value that is not null, and not an empty string. MapMessage inherits from the Message interface and adds a message body that contains a Map.

The primitive types can be read or written explicitly using methods for each type. They may also be read or written generically as objects. For instance, a call to MapMessage.setInt("foo", 6) is equivalent to MapMessage.setObject("foo", new Integer(6)).

When a client receives a MapMessage, it is in read-only mode. If a client attempts to write to the message at this point, a MessageNotWriteableException is thrown. If clearBody is called, the message can now be both read from and written to.

MapMessage objects support conversions (see table below). Unsupported conversions must throw a JMSEException. The String-to-primitive conversions may throw a runtime exception if the primitive's valueOf() method does not accept it as a valid String representation of the primitive.

A value written as the row type can be read as the column type.

	boolean	byte	short	char	int	long	float	double	String	byte[]
boolean	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	X
String	X	X	X	X	X	X
byte[]						X

Attempting to read a null value as a primitive type must be treated as calling the primitive's corresponding `valueOf(String)` conversion method with a null value. Since `char` does not support a String conversion, attempting to read a null value as a `char` must throw a `NullPointerException`.

Member Function Documentation

void MapMessage.clearBody () throws JMSException

API method. Clears out the message body.

Calling this method leaves the message body in the same state as an empty body in a newly created message.

Exceptions:

JMSException	Actually never thrown.
--------------	------------------------

boolean MapMessage.getBoolean (String name) throws JMSException

API method. Returns the boolean value to which the specified key is mapped.

Parameters:

name	the key whose associated value is to be returned.
------	---

Returns:

the boolean value associated with the specified name.

Exceptions:

MessageFormatException	If the value type is invalid.
------------------------	-------------------------------

byte MapMessage.getByte (String name) throws JMSException

API method. Returns the byte value to which the specified key is mapped.

Parameters:

name	the key whose associated value is to be returned.
------	---

Returns:

the byte value associated with the specified name.

Exceptions:

MessageFormatException	If the value type is invalid.
------------------------	-------------------------------

byte [] MapMessage.getBytes (String name) throws JMSException

API method. Returns the byte array value to which the specified key is mapped.

Parameters:

name	the key whose associated value is to be returned.
------	---

Returns:

a copy of the byte array value with the specified name; if there is no item by this name, a null value is returned.

Exceptions:

MessageFormatException	If the value type is invalid.
------------------------	-------------------------------

char MapMessage.getChar (String name) throws JMSEException

API method. Returns the char value to which the specified key is mapped.

Parameters:

name	the key whose associated value is to be returned.
------	---

Returns:

the char value associated with the specified name.

Exceptions:

MessageFormatException	If the value type is invalid.
------------------------	-------------------------------

double MapMessage.getDouble (String name) throws JMSEException

API method. Returns the double value to which the specified key is mapped.

Parameters:

name	the key whose associated value is to be returned.
------	---

Returns:

the double value associated with the specified name.

Exceptions:

MessageFormatException	If the value type is invalid.
------------------------	-------------------------------

float MapMessage.getFloat (String name) throws JMSEException

API method. Returns the float value to which the specified key is mapped.

Parameters:

name	the key whose associated value is to be returned.
------	---

Returns:

the float value associated with the specified name.

Exceptions:

MessageFormatException	If the value type is invalid.
------------------------	-------------------------------

int MapMessage.getInt (String name) throws JMSEException

API method. Returns the int value to which the specified key is mapped.

Parameters:

name	the key whose associated value is to be returned.
------	---

Returns:

the int value associated with the specified name.

Exceptions:

MessageFormatException	If the value type is invalid.
------------------------	-------------------------------

long MapMessage.getLong (String name) throws JMSEException

API method. Returns the long value to which the specified key is mapped.

Parameters:

name	the key whose associated value is to be returned.
------	---

Returns:

the long value associated with the specified name.

Exceptions:

MessageFormatException	If the value type is invalid.
------------------------	-------------------------------

Enumeration MapMessage.getMapNames () throws JMSEException

API method. Returns an Enumeration of all the names in this MapMessage.

Returns:

an enumeration of all the names in this MapMessage.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

Object MapMessage.getObject (String name) throws JMSEException

API method. This method returns in objectified format a java primitive type that had been stored in the Map with the equivalent setObject method call, or its equivalent primitive settype method.

Parameters:

name	the key whose associated value is to be returned.
------	---

Returns:

a copy of the Java object value with the specified name, in objectified format (for example, if the object was set as an int, an Integer is returned); if there is no item by this name, a null value is returned.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

short MapMessage.getShort (String name) throws JMSEException

API method. Returns the short value to which the specified key is mapped.

Parameters:

name	the key whose associated value is to be returned.
------	---

Returns:

the short value associated with the specified name.

Exceptions:

MessageFormatException	If the value type is invalid.
------------------------	-------------------------------

String MapMessage.getString (String name) throws JMSEException

API method. Returns the String value to which the specified key is mapped.

Parameters:

name	the key whose associated value is to be returned.
------	---

Returns:

the String value associated with the specified name; if there is no item by this name, a null value is returned.

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

boolean MapMessage.itemExists (String name) throws JMSEException

API method. Indicates whether an item exists in this MapMessage.

Parameters:

name	the name of the item to test.
------	-------------------------------

Returns:

true if the item exists

Exceptions:

JMSEException	Actually never thrown.
---------------	------------------------

void MapMessage.setBoolean (String name, boolean value) throws JMSException

API method. Sets a boolean value with the specified name into the Map.

Parameters:

name	the name of the boolean.
value	the boolean value to set in the Map

Exceptions:

MessageNotWritableException	If the message body is read-only.
-----------------------------	-----------------------------------

void MapMessage.setByte (String name, byte value) throws JMSException

API method. Sets a byte value with the specified name into the Map.

Parameters:

name	the name of the byte.
value	the value to set in the Map

Exceptions:

MessageNotWritableException	If the message body is read-only.
-----------------------------	-----------------------------------

void MapMessage.setBytes (String name, byte[] value) throws JMSException

API method. Sets a byte array value with the specified name into the Map.

Parameters:

name	the name of the byte array.
value	the byte array value to set in the Map; the array is copied so that the value for name will not be altered by future modifications

Exceptions:

MessageNotWritableException	If the message body is read-only.
-----------------------------	-----------------------------------

void MapMessage.setBytes (String name, byte[] value, int offset, int length) throws JMSException

API method. Sets a portion of a byte array value with the specified name into the Map.

Parameters:

name	the name of the byte array.
value	the byte array value to set in the Map; the array is copied so that the value for name will not be altered by future modifications.
offset	the initial offset within the byte array.
length	the number of bytes to use.

Exceptions:

MessageNotWritableException	If the message body is read-only.
-----------------------------	-----------------------------------

void MapMessage.setChar (String name, char value) throws JMSException

API method. Sets a char value with the specified name into the Map.

Parameters:

name	the name of the char.
value	the value to set in the Map

Exceptions:

MessageNotWritableException	If the message body is read-only.
-----------------------------	-----------------------------------

void MapMessage.setDouble (String name, double value) throws JMSException

API method. Sets a double value with the specified name into the Map.

Parameters:

name	the name of the double.
value	the value to set in the Map

Exceptions:

MessageNotWritableException	If the message body is read-only.
on	

void MapMessage.setFloat (String name, float value) throws JMSException

API method. Sets a float value with the specified name into the Map.

Parameters:

name	the name of the float.
value	the value to set in the Map

Exceptions:

MessageNotWritableException	If the message body is read-only.
on	

void MapMessage.setInt (String name, int value) throws JMSException

API method. Sets an int value with the specified name into the Map.

Parameters:

name	the name of the int.
value	the value to set in the Map

Exceptions:

MessageNotWritableException	If the message body is read-only.
on	

void MapMessage.setLong (String name, long value) throws JMSException

API method. Sets a long value with the specified name into the Map.

Parameters:

name	the name of the long.
value	the value to set in the Map

Exceptions:

MessageNotWritableException	If the message body is read-only.
on	

void MapMessage.setObject (String name, Object value) throws JMSException

API method. Sets an object value with the specified name into the Map.

This method works only for the objectified primitive object types (Integer, Double, Long ...), String objects, and byte arrays.

Parameters:

name	the name of the object.
value	the Java object value to set in the Map

Exceptions:

MessageNotWritableException	If the message body is read-only.
MessageFormatException	If the value type is invalid.

void MapMessage.setShort (String name, short value) throws JMSException

API method. Sets a short value with the specified name into the Map.

Parameters:

name	the name of the short.
value	the value to set in the Map

Exceptions:

MessageNotWritableException	If the message body is read-only.
-----------------------------	-----------------------------------

void MapMessage.setString (String name, String value) throws JMSException

API method. Sets a String value with the specified name into the Map.

Parameters:

name	the name of the String.
value	the value to set in the Map

Exceptions:

MessageNotWritableException	If the message body is read-only.
-----------------------------	-----------------------------------

5.4. StreamMessage

Class org.objectweb.joram.client.jms.StreamMessage
 extends org.objectweb.joram.client.jms.Message
 implements javax.jms.StreamMessage

Public Member Functions

```
void clearBody () throws JMSException
void writeBoolean (boolean value) throws JMSException
void writeByte (byte value) throws JMSException
void writeBytes (byte[] value) throws JMSException
void writeBytes (byte[] value, int offset, int length) throws JMSException
void writeChar (char value) throws JMSException
void writeDouble (double value) throws JMSException
void writeFloat (float value) throws JMSException
void writeInt (int value) throws JMSException
void writeLong (long value) throws JMSException
void writeShort (short value) throws JMSException
void writeString (String value) throws JMSException
void writeObject (Object value) throws JMSException
boolean readBoolean () throws JMSException
byte readByte () throws JMSException
short readShort () throws JMSException
char readChar () throws JMSException
int readInt () throws JMSException
long readLong () throws JMSException
float readFloat () throws JMSException
double readDouble () throws JMSException
int readBytes (byte[] bytes) throws JMSException
String readString () throws JMSException
Object readObject () throws JMSException
void reset () throws JMSException
```

Detailed Description

Implements the javax.jms.StreamMessage interface.

A StreamMessage object is used to send a stream of primitive Java types. It is filled and read sequentially. It inherits from the Message interface and adds a stream message body. Its methods are based largely on those found in java.io.DataInputStream and java.io.DataOutputStream.

The primitive types can be read or written explicitly using methods for each type. They may also be read or written generically as objects. For instance, a call to StreamMessage.writeInt(6) is equivalent to StreamMessage.writeObject(new Integer(6)).

When the message is first created, and when clearBody is called, the body of the message is in write-only mode. After the first call to reset has been made, the message body is in read-only mode. After a message has been sent, the client that sent it can retain and modify it without affecting the message that has been sent. The same message object can be sent multiple times. When a message has been received, the provider has called reset so that the message body is in read-only mode for the client.

If clearBody is called on a message in read-only mode, the message body is cleared and the message body is in write-only mode.

If a client attempts to read a message in write-only mode, a MessageNotReadableException is thrown.

If a client attempts to write a message in read-only mode, a MessageNotWriteableException is thrown.

StreamMessage objects support conversions (see table below). Unsupported conversions must throw a JMSEException. The String-to-primitive conversions may throw a runtime exception if the primitive's valueOf() method does not accept it as a valid String representation of the primitive.

A value written as the row type can be read as the column type.

	boolean	byte	short	char	int	long	float	double	String	byte[]
boolean	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

Attempting to read a null value as a primitive type must be treated as calling the primitive's corresponding valueOf(String) conversion method with a null value. Since char does not support a String conversion, attempting to read a null value as a char must throw a NullPointerException.

Member Function Documentation

void StreamMessage.clearBody () throws JMSEException

API method. Clears out the message body.

Calling this method leaves the message body in the same state as an empty body in a newly created message.

Exceptions:

JMSEException	In case of an error while closing the input or output streams.
---------------	--

boolean StreamMessage.readBoolean () throws JMSEException

API method. Reads a boolean from the StreamMessage.

Returns:

the value read.

Exceptions:

MessageNotReadableExcept	If the message body is write-only.
--------------------------	------------------------------------

ion	
MessageFormatException	If reading the expected type is not possible.
MessageEOFException	Unexpected end of bytes array.
JMSEException	internal error

byte StreamMessage.readByte () throws JMSEException

API method. Reads a byte from the StreamMessage.

Returns:

the value read.

Exceptions:

MessageNotReadableExcept ion	If the message body is write-only.
MessageFormatException	If reading the expected type is not possible.
MessageEOFException	Unexpected end of bytes array.
JMSEException	internal error

int StreamMessage.readBytes (byte[] bytes) throws JMSEException

API method. Reads a byte array from the StreamMessage into the specified buffer.

To read the field value, readBytes should be successively called until it returns a value less than the length of the read buffer. The value of the bytes in the buffer following the last byte read is undefined. If readBytes returns a value equal to the length of the buffer, a subsequent readBytes call must be made. If there are no more bytes to be read, this call returns -1.

If the byte array field value is null, readBytes returns -1.

If the byte array field value is empty, readBytes returns 0.

Once the first readBytes call on a byte[] field value has been made, the full value of the field must be read before it is valid to read the next field. An attempt to read the next field before that has been done will throw a MessageFormatException.

To read the byte field value into a new byte[] object, use the readObject method.

Parameters:

bytes	the buffer into which the data is read.
-------	---

Returns:

the total number of bytes read into the buffer, or -1 if there is no more data because the end of the byte field has been reached.

Exceptions:

MessageNotReadableExcept ion	If the message body is write-only.
MessageFormatException	If reading the expected type is not possible.
MessageEOFException	Unexpected end of bytes array.
JMSEException	internal error

char StreamMessage.readChar () throws JMSEException

API method. Reads a byte from the StreamMessage.

Returns:

the value read.

Exceptions:

MessageNotReadableExcept ion	If the message body is write-only.
MessageFormatException	If reading the expected type is not possible.
MessageEOFException	Unexpected end of bytes array.
JMSEException	internal error

double StreamMessage.readDouble () throws JMSEException

API method. Reads a double from the StreamMessage.

Returns:

the value read.

Exceptions:

MessageNotReadableException	If the message body is write-only.
MessageFormatException	If reading the expected type is not possible.
MessageEOFException	Unexpected end of bytes array.
JMSEException	internal error

float StreamMessage.readFloat () throws JMSEException

API method. Reads a float from the StreamMessage.

Returns:

the value read.

Exceptions:

MessageNotReadableException	If the message body is write-only.
MessageFormatException	If reading the expected type is not possible.
MessageEOFException	Unexpected end of bytes array.
JMSEException	internal error

int StreamMessage.readInt () throws JMSEException

API method. Reads an int from the StreamMessage.

Returns:

the value read.

Exceptions:

MessageNotReadableException	If the message body is write-only.
MessageFormatException	If reading the expected type is not possible.
MessageEOFException	Unexpected end of bytes array.
JMSEException	internal error

long StreamMessage.readLong () throws JMSEException

API method. Reads a long from the StreamMessage.

Returns:

the value read.

Exceptions:

MessageNotReadableException	If the message body is write-only.
MessageFormatException	If reading the expected type is not possible.
MessageEOFException	Unexpected end of bytes array.
JMSEException	internal error

Object StreamMessage.readObject () throws JMSEException

API method. Reads an objectified primitive type from the StreamMessage.

Returns:

the value read.

Exceptions:

MessageNotReadableException	If the message body is write-only.
MessageFormatException	If reading the body is not possible.
MessageEOFException	Unexpected end of bytes array.
JMSEException	internal error

Short StreamMessage.readShort () throws JMSEException
API method. Reads a short from the StreamMessage.

Returns:

the value read.

Exceptions:

MessageNotReadableException	If the message body is write-only.
MessageFormatException	If reading the expected type is not possible.
MessageEOFException	Unexpected end of bytes array.
JMSEException	internal error

String StreamMessage.readString () throws JMSEException

API method. Reads a String from the StreamMessage.

Returns:

a Unicode string from the StreamMessage.

Exceptions:

MessageNotReadableException	If the message body is write-only.
MessageFormatException	If reading the expected type is not possible.
MessageEOFException	Unexpected end of bytes array.
JMSEException	internal error

void StreamMessage.reset () throws JMSEException

API method. Puts the message body in read-only mode and reset the stream to the beginning.

Exceptions:

JMSEException	If an error occurs while closing the output stream.
---------------	---

void StreamMessage.writeBoolean (boolean value) throws JMSEException

API method. Writes a boolean to the stream message as a 1-byte value. The value true is written as the value 1; the value false is written as the value 0.

Parameters:

value	the value to be written.
-------	--------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSEException	If the value could not be written on the stream.

void StreamMessage.writeByte (byte value) throws JMSEException

API method. Writes a byte to the stream message.

Parameters:

value	the value to be written.
-------	--------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
-----------------------------	-----------------------------------

JMSEException	If the value could not be written on the stream.
---------------	--

void StreamMessage.writeBytes (byte[] value) throws JMSEException

API method. Writes a byte array to the StreamMessage.

Each byte array is written to the message as a separate byte array field. Consecutively written byte array fields are treated as distinct fields when the fields are read.

Parameters:

value	the byte array to be written.
-------	-------------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSEException	If the value could not be written on the stream.

void StreamMessage.writeBytes (byte[] value, int offset, int length) throws JMSEException

API method. Writes a portion of a byte array to the StreamMessage.

The portion of the byte array is written to the message as a separate byte array field. Consecutively written byte array fields are treated as distinct fields when the fields are read.

Parameters:

value	the byte array to be written.
offset	the initial offset within the byte array
length	the number of bytes to use

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSEException	If the value could not be written on the stream.

void StreamMessage.writeChar (char value) throws JMSEException

API method. Writes a char to the stream message.

Parameters:

value	the value to be written.
-------	--------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSEException	If the value could not be written on the stream.

void StreamMessage.writeDouble (double value) throws JMSEException

API method. Writes a double to the stream message.

Parameters:

value	the value to be written.
-------	--------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSEException	If the value could not be written on the stream.

void StreamMessage.writeFloat (float value) throws JMSEException

API method. Writes a float to the stream message.

Parameters:

value	the value to be written.
-------	--------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSEException	If the value could not be written on the stream.

void StreamMessage.writeInt (int value) throws JMSEException

API method. Writes an int to the stream message.

Parameters:

value	the value to be written.
-------	--------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSEException	If the value could not be written on the stream.

void StreamMessage.writeLong (long value) throws JMSEException

API method. Writes a long to the stream message.

Parameters:

value	the value to be written.
-------	--------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSEException	If the value could not be written on the stream.

void StreamMessage.writeObject (Object value) throws JMSEException

API method. Writes an object to the stream message. This method works only for the objectified primitive object types (Integer, Double, Long ...), String objects, and byte arrays.

Parameters:

value	the value to be written.
-------	--------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
MessageFormatException	If the value type is invalid.
JMSEException	If the value could not be written on the stream.

void StreamMessage.writeShort (short value) throws JMSEException

API method. Writes a short to the stream message.

Parameters:

value	the value to be written.
-------	--------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSEException	If the value could not be written on the stream.

void StreamMessage.writeString (String value) throws JMSEException

API method. Writes a String to the stream message.

Parameters:

value	the value to be written.
-------	--------------------------

Exceptions:

MessageNotWritableException	If the message body is read-only.
JMSException	If the value could not be written on the stream.

5.5. ObjectMessage

Class org.objectweb.joram.client.jms.ObjectMessage
 extends org.objectweb.joram.client.jms.Message
 implements javax.jms.ObjectMessage

Public Member Functions

void setObject (Serializable obj) throws JMSException
 Serializable getObject () throws MessageFormatException

Detailed Description

Implements the javax.jms.ObjectMessage interface.

An ObjectMessage object is used to send a message that contains a unique serializable Java object. It inherits from the Message interface and adds a body containing a single reference to an object.

When a client receives an ObjectMessage, it is in read-only mode. If a client attempts to write to the message at this point, a MessageNotWritableException is thrown. If clearBody is called, the message can now be both read from and written to.

Member Function Documentation**Serializable ObjectMessage.getObject () throws MessageFormatException**

API method. Gets the serializable object containing this message's data, null if none.

Returns:

the serializable object containing this message's data.

Exceptions:

MessageFormatException	In case of a problem when getting the body.
------------------------	---

void ObjectMessage.setObject (Serializable obj) throws JMSException

API method. Sets the serializable object containing this message's data. It is important to note that an ObjectMessage contains a snapshot of the object at the time setObject() is called; subsequent modifications of the object will have no effect on the ObjectMessage body.

Parameters:

object	the message's data.
--------	---------------------

Exceptions:

MessageNotWritableException	When trying to set an object if the message body is read-only.
MessageFormatException	If object serialization fails.

5.6. TextMessage

Class org.objectweb.joram.client.jms.TextMessage
 extends org.objectweb.joram.client.jms.Message
 implements javax.jms.TextMessage

Public Member Functions

`void setText (String text) throws MessageNotWriteableException, MessageFormatException`
`String getText () throws JMSException`

Detailed Description

Implements the javax.jms.TextMessage interface.

A TextMessage object is used to send a message containing a String. It inherits from the Message interface and adds a text message body.

This message type can be used to transport text-based messages, including those with XML content.

When a client receives a TextMessage, it is in read-only mode. If a client attempts to write to the message at this point, a MessageNotWriteableException is thrown. If clearBody is called, the message can now be both read from and written to.

Member Function Documentation

String TextMessage.getText () throws JMSException

API method. Returns the text body of the message, null if none.

Returns:

the String containing this message's data.

Exceptions:

JMSException	In case of a problem when getting the body.
--------------	---

void TextMessage.setText (String text) throws MessageNotWriteableException, MessageFormatException

API method. Sets a String as the body of the message.

Parameters:

text	the String containing the message's data.
------	---

Exceptions:

MessageNotWriteableException	When trying to set the text if the message body is read-only.
MessageFormatException	If the text serialization fails.

6. Destination API

Class Reference

This chapter describes classes allowing the creation of specialized destinations like JMS acquisition or distribution queue or topic. These classes are simple factories to simplify the declaration of these complex objects.

6.1. JMSAcquisitionQueue

Class org.objectweb.joram.client.jms.admin.JMSAcquisitionQueue

Static Public Member Functions

```
static Queue create (String dest) throws ConnectException, AdminException
static Queue create (int serverId, String dest) throws ConnectException, AdminException
static Queue create (int serverId, String name, String dest) throws ConnectException, AdminException
static Queue create (int serverId, String name, String dest, Properties props) throws ConnectException, AdminException
```

Static Public Attributes

```
static final String JMSAcquisition = "org.objectweb.joram.mom.dest.jms.JMSAcquisition"
```

Detailed Description

The JMSAcquisitionQueue class allows administrators to create JMS acquisition queues (JMS bridge in). The JMS bridge destinations rely on a particular Joram service which purpose is to maintain valid connections with the foreign XMQ servers.

Member Function Documentation

static Queue JMSAcquisitionQueue.create (String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMS acquisition queue on the local server.

The request fails if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

dest	The name of the foreign destination.
------	--------------------------------------

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
------------------	---

AdminException	If the request fails.
----------------	-----------------------

static Queue JMSAcquisitionQueue.create (int serverId, String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMS acquisition queue on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
dest	The name of the foreign destination.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Queue JMSAcquisitionQueue.create (int serverId, String name, String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMS acquisition queue on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.
dest	The name of the foreign destination.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Queue JMSAcquisitionQueue.create (int serverId, String name, String dest, Properties props) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMS acquisition queue on a given server.

A set of properties is used to configure the distribution destination:

- period – .
- acquisition.period - The period between two acquisitions, default is 0 (no periodic acquisition).
- persistent - Tells if produced messages will be persistent, default is true (JMS default).
- expiration - Tells the life expectancy of produced messages, default is 0 (JMS default time to live).
- priority - Tells the JMS priority of produced messages, default is 4 (JMS default).
- acquisition.max_msg - The maximum number of messages between the last message acquired by the handler and the message correctly handled by the acquisition destination, default is 20. When the number of messages waiting to be handled is greater the acquisition handler is temporarily stopped. A value lesser or equal to 0 disables the mechanism.
- acquisition.min_msg - The minimum number of message to restart the acquisition, default is 10.
- acquisition.max_pnd - The maximum number of pending messages on the acquisition destination, default is 20. When the number of waiting messages is greater the acquisition handler is temporarily stopped. A value lesser or equal to 0 disables the mechanism.
- acquisition.min_pnd - The minimum number of pending messages to restart the acquisition, default is 10.

- jms.ConnectionUpdatePeriod - Period between two update phases allowing the discovering of new Connections.
- jms.DurableSubscriptionName - If the XMQ destination is a topic, this property sets the name of the durable subscription created. If absent, the subscription will not be durable and messages published when connection with XMQ server is failing will be lost.
- jms.Selector - Expression used for filtering messages from the XMQ destination.
- jms.Routing - This property allows to filter the connections used to acquire messages.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.
dest	The name of the foreign destination.
props	A Properties object containing all needed parameters.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

Member Data Documentation

final String JMSAcquisitionQueue.JMSAcquisition "org.objectweb.joram.mom.dest.jms.JMSAcquisition" [static] =

Class name of handler allowing to acquire messages to a foreign JMS provider.

6.2. JMSAcquisitionTopic

Class org.objectweb.joram.client.jms.admin.JMSAcquisitionTopic

Static Public Member Functions

```
static Topic create (String dest) throws ConnectException, AdminException
static Topic create (int serverId, String dest) throws ConnectException, AdminException
static Topic create (int serverId, String name, String dest) throws ConnectException, AdminException
static Topic create (int serverId, String name, String dest, Properties props) throws ConnectException, AdminException
```

Static Public Attributes

static final String JMSAcquisition = "org.objectweb.joram.mom.dest.jms.JMSAcquisition"

Detailed Description

The JMSAcquisitionTopic class allows administrators to create JMS acquisition topics (JMS bridge in). The JMS bridge destinations rely on a particular Joram service which purpose is to maintain valid connections with the foreign XMQ servers.

Member Function Documentation

static Topic JMSAcquisitionTopic.create (String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMS acquisition topic on the local server.

The request fails if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

dest	The name of the foreign destination.
------	--------------------------------------

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Topic JMSAcquisitionTopic.create (int serverId, String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMS acquisition topic on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
dest	The name of the foreign destination.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Topic JMSAcquisitionTopic.create (int serverId, String name, String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMS acquisition topic on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the created topic.
dest	The name of the foreign destination.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Topic JMSAcquisitionTopic.create (int serverId, String name, String dest, Properties props) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMS acquisition topic on a given server.

A set of properties is used to configure the distribution destination:

- period – .
- acquisition.period - The period between two acquisitions, default is 0 (no periodic acquisition).
- persistent - Tells if produced messages will be persistent, default is true (JMS default).
- expiration - Tells the life expectancy of produced messages, default is 0 (JMS default time to live).
- priority - Tells the JMS priority of produced messages, default is 4 (JMS default).
- jms.ConnectionUpdatePeriod - Period between two update phases allowing the discovering of new Connections.
- jms.DurableSubscriptionName - If the XMQ destination is a topic, this property sets the name of the durable subscription created. If absent, the subscription will not be durable and messages published when connection with XMQ server is failing will be lost.
- jms.Selector - Expression used for filtering messages from the XMQ destination.
- jms.Routing - This property allows to filter the connections used to acquire messages.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the created topic.
dest	The name of the foreign destination.
props	A Properties object containing all needed parameters.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

Member Data Documentation

final String JMSAcquisitionTopic.JMSAcquisition "org.objectweb.joram.mom.dest.jms.JMSAcquisition" [static] =

Class name of handler allowing to acquire messages to a foreign JMS provider.

6.3. JMSDistributionQueue

Class org.objectweb.joram.client.jms.admin.JMSDistributionQueue

Static Public Member Functions

static Queue create (String dest) throws ConnectException, AdminException

static Queue create (int serverId, String dest) throws ConnectException, AdminException

static Queue create (int serverId, String name, String dest) throws ConnectException, AdminException

static Queue create (int serverId, String name, String dest, Properties props) throws ConnectException, AdminException

Static Public Attributes

static final String JMSDistribution = "org.objectweb.joram.mom.dest.jms.JMSDistribution"

Detailed Description

The `JMSDistributionQueue` class allows administrators to create JMS distribution queues (JMS bridge out). The JMS bridge destinations rely on a particular Joram service which purpose is to maintain valid connections with the foreign XMQ servers.

Member Function Documentation

static Queue JMSDistributionQueue.create (String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMS distribution queue on the local server.

The request fails if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

dest	The name of the foreign destination.
------	--------------------------------------

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

See also:

`create(int, String, String, Properties)`

static Queue JMSDistributionQueue.create (int serverId, String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMS distribution queue on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
dest	The name of the foreign destination.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

See also:

`create(int, String, String, Properties)`

static Queue JMSDistributionQueue.create (int serverId, String name, String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMS distribution queue on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
----------	---

name	The name of the created queue.
dest	The name of the foreign destination.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

See also:

`create(int, String, String, Properties)`

static Queue JMSDistributionQueue.create (int serverId, String name, String dest, Properties props) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMS distribution queue on a given server.

A set of properties is used to configure the distribution destination:

- period – Tells the time to wait before another distribution attempt. Default is 0, which means there won't be other attempts.
- distribution.batch – If set to true, the destination will try to distribute each time every waiting message, regardless of distribution errors. This can lead to the loss of message ordering, but will prevent a blocking message from blocking every following message. When set to false, the distribution process will stop on the first error. Default is false.
- distribution.async - If set to true, the messages are asynchronously forwarded through a daemon.
- jms.ConnectionUpdatePeriod - Period between two update phases allowing the discovering of new Connections.
- jms.Routing - This property allows to filter the connections used to forward the messages. It can be set either globally at creation or specifically for a message at sending.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.
dest	The name of the foreign destination.
props	A Properties object containing all needed parameters.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

Member Data Documentation**final String JMSDistributionQueue.JMSDistribution
"org.objectweb.joram.mom.dest.jms.JMSDistribution" [static]** =

Class name of handler allowing to distribute messages to a foreign JMS provider.

6.4. JMSDistributionTopic

Class org.objectweb.joram.client.jms.admin.JMSDistributionTopic

Static Public Member Functions

```
static Topic create (String dest) throws ConnectException, AdminException
static Topic create (int serverId, String dest) throws ConnectException, AdminException
static Topic create (int serverId, String name, String dest) throws ConnectException, AdminException
static Topic create (int serverId, String name, String dest, Properties props) throws ConnectException, AdminException
```

Static Public Attributes

```
static final String JMSDistribution = "org.objectweb.joram.mom.dest.jms.JMSDistribution"
```

Detailed Description

The JMSDistributionTopic class allows administrators to create JMS distribution topics (JMS bridge out). The JMS bridge destinations rely on a particular Joram service which purpose is to maintain valid connections with the foreign XMQ servers.

Member Function Documentation

static Topic JMSDistributionTopic.create (String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMS distribution topic on the local server. The request fails if the destination deployment fails server side. Be careful this method use the static AdminModule connection.

Parameters:

dest	The name of the foreign destination.
------	--------------------------------------

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

See also:

create(int, String, String, Properties)

static Topic JMSDistributionTopic.create (int serverId, String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMS distribution topic on a given server. The request fails if the target server does not belong to the platform, or if the destination deployment fails server side. Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
dest	The name of the foreign destination.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

See also:

`create(int, String, String, Properties)`

static Topic JMSDistributionTopic.create (int serverId, String name, String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMS distribution topic on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the created topic.
dest	The name of the foreign destination.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

See also:

`create(int, String, String, Properties)`

static Topic JMSDistributionTopic.create (int serverId, String name, String dest, Properties props) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMS distribution topic on a given server.

A set of properties is used to configure the distribution destination:

- period – Tells the time to wait before another distribution attempt. Default is 0, which means there won't be other attempts.
- distribution.batch – If set to true, the destination will try to distribute each time every waiting message, regardless of distribution errors. This can lead to the loss of message ordering, but will prevent a blocking message from blocking every following message. When set to false, the distribution process will stop on the first error. Default is false.
- distribution.async - If set to true, the messages are asynchronously forwarded through a daemon.
- jms.ConnectionUpdatePeriod - Period between two update phases allowing the discovering of new Connections.
- jms.Routing - This property allows to filter the connections used to forward the messages. It can be set either globally at creation or specifically for a message at sending.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the created topic.
dest	The name of the foreign destination.
props	A Properties object containing all needed parameters.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

Member Data Documentation

final String JMSDistributionTopic.JMSDistribution = "org.objectweb.joram.mom.dest.jms.JMSDistribution" [static]

Class name of handler allowing to distribute messages to a foreign JMS provider.

6.5. RestAcquisitionQueue

Class org.objectweb.joram.client.jms.admin.RestAcquisitionQueue

Public Member Functions

```

String getHost ()
RestAcquisitionQueue setHost (String host)
int getPort ()
RestAcquisitionQueue setPort (int port)
boolean isUseOldAPI()
RestAcquisitionQueue setUseOldAPI(boolean useOldAPI)
String getUsername ()
RestAcquisitionQueue setUsername (String userName)
String getPassword ()
RestAcquisitionQueue setPassword (String password)
boolean isMediaTypeJson ()
RestAcquisitionQueue setMediaTypeJson (boolean mediaTypeJson)
long getTimeout ()
RestAcquisitionQueue setTimeout (long timeout)
long getAcquisitionPeriod ()
RestAcquisitionQueue setAcquisitionPeriod (int acquisitionPeriod)
Queue create (String dest) throws ConnectException, AdminException
Queue create (int serverId, String dest) throws ConnectException, AdminException
Queue create (int serverId, String name, String dest) throws ConnectException, AdminException
Queue create (int serverId, String name, String dest, Properties props) throws ConnectException, AdminException

```

Static Public Attributes

static final String RESTAcquisition = "com.scalagent.joram.mom.dest.rest.RestAcquisitionAsync"

Detailed Description

The **RestAcquisitionQueue** class allows administrators to create REST acquisition queues (Rest bridge in). The Rest bridge destinations rely on a particular Joram service which purpose is to maintain valid connections with the foreign REST Joram servers.

The valid properties define by this destination are:

- rest.host: hostname or IP address of remote JMS REST server, by default "localhost".
- rest.port: listening port of remote JMS REST server, by default 8989.
- rest.useOldAPI: if true, use the old Rest/JMS API with authentication through query parameters.
- rest.user: user name needed to connect to remote JMS REST server, by default "anonymous".
- rest.pass: password needed to connect to remote JMS REST server, by default "anonymous".
- rest.mediaTypeJson: if true the acquisition queue accepts JSON message, default true.
- rest.timeout: Timeout for waiting for a message, by default 10.000 (10 seconds).
- rest.idletimeout: normally each remote resource need to be explicitly closed, this parameter allows to set the idle time in seconds in which the remote context will be closed if idle. If less than or equal to 0, the context never closes. By default 60 seconds.
- rest.checkPeriod: verification period in seconds of acquisition handler. If no message has been received during this period, the handler is restarted.

- acquisition.period: delay between attempts to receive a message if the acquisition queue doesn't use the synchronous mode, by default 100 milliseconds.
- jms.destination: the name of remote JMS destination.

Member Function Documentation

String org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.getHost ()

Returns:

the hostName

RestAcquisitionQueue org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.setHost (String host)

Parameters:

<i>host</i>	the hostname or IP address of the remote Rest / JMS service
-------------	---

int org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.getPort ()

Returns:

the port

RestAcquisitionQueue org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.setPort (int port)

Parameters:

<i>port</i>	the listening port of the remote Rest / JMS service
-------------	---

boolean org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.isUseOldAPI()

Returns:

value of the option useOldAPI.

RestAcquisitionQueue org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.setUseOldAPI (boolean useOldAPI)

Parameters:

<i>useOldAPI</i>	the value of the option useOldAPI.
------------------	------------------------------------

String org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.getUsername ()

Returns:

the userName

RestAcquisitionQueue org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.setUsername (String userName)

Parameters:

<i>userName</i>	the userName to set
-----------------	---------------------

String org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.getPassword ()

Returns:

the password

RestAcquisitionQueue

org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.setPassword (String password)

Parameters:

<i>password</i>	the password to set
-----------------	---------------------

boolean org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.isMediaTypeJson ()

Returns:

true if the consumer interprets the message as Json.

RestAcquisitionQueue

org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.setMediaTypeJson (boolean mediaTypeJson)

Parameters:

<i>mediaTypeJson</i>	Set to true if the consumer must interpret the message as Json.
----------------------	---

long org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.getTimeout ()

Returns:

the timeout

RestAcquisitionQueue **org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.setTimeout (long timeout)**

Parameters:

<i>timeout</i>	the timeout to set
----------------	--------------------

long org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.getAcquisitionPeriod ()

Returns:

the acquisitionPeriod

RestAcquisitionQueue

org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.setAcquisitionPeriod (int acquisitionPeriod)

Parameters:

<i>acquisitionPeriod</i>	the acquisitionPeriod to set
--------------------------	------------------------------

Queue org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.create (String dest) throws ConnectException, AdminException

Administration method creating and deploying a REST acquisition queue on the local server.
The request fails if the destination deployment fails server side.

Be careful this method use the static **AdminModule** connection.

Parameters:

dest	The name of the foreign destination.
------	--------------------------------------

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

See also:

`create(int, String, String, Properties)`

Queue `org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.create (int serverId, String dest) throws ConnectException, AdminException`

Administration method creating and deploying a REST acquisition queue on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static **AdminModule** connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
dest	The name of the foreign destination.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

See also:

`create(int, String, String, Properties)`

Queue `org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.create (int serverId, String name, String dest) throws ConnectException, AdminException`

Administration method creating and deploying a REST acquisition queue on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static **AdminModule** connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.
dest	The name of the foreign destination.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

See also:

`create(int, String, String, Properties)`

Queue `org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.create (int serverId, String name, String dest, Properties props) throws ConnectException, AdminException`

Administration method creating and deploying a REST acquisition queue on a given server.

In addition to properties used to configure acquisition queues a set of specific properties allows to configure Rest/JMS acquisition destination:

- rest.host: hostname or IP address of remote JMS REST server, by default "localhost".
- rest.port: listening port of remote JMS REST server, by default 8989.
- rest.user: user name needed to connect to remote JMS REST server, by default "anonymous".
- rest.pass: password needed to connect to remote JMS REST server, by default "anonymous".
- rest.mediaTypeJson: if true the acquisition queue accepts JSON message, default true.
- rest.timeout: Timeout for waiting for a message, by default 10.000 (10 seconds).
- rest.idletimeout: normally each remote resource need to be explicitly closed, this parameter allows to set the idle time in seconds in which the remote context will be closed if idle. If less than or equal to 0, the context never closes. By default 60 seconds.
- acquisition.period: delay between attempts to receive a message if the acquisition queue doesn't use the asynchronous mode, by default 100 milliseconds.
- jms.destination: the name of remote JMS destination.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static **AdminModule** connection.

Parameters:

<code>serverId</code>	The identifier of the server where deploying the queue.
<code>name</code>	The name of the created queue.
<code>dest</code>	The name of the foreign destination.
<code>props</code>	A Properties object containing all needed parameters.

Returns:

the created bridge destination.

Exceptions:

<code>ConnectException</code>	If the administration connection is closed or broken.
<code>AdminException</code>	If the request fails.

Member Data Documentation

```
final String org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.RESTAcquisition =
"com.scalagent.joram.mom.dest.rest.RestAcquisitionAsync" [static]
```

Class name of handler allowing to acquire messages to a foreign REST provider.

6.6. *RestDistributionQueue*

Class `org.objectweb.joram.client.jms.admin.RestDistributionQueue`

Public Member Functions

```
String getHost ()
RestDistributionQueue setHost (String host)
int getPort ()
RestDistributionQueue setPort (int port)
boolean isUseOldAPI()
RestAcquisitionQueue setUseOldAPI(boolean useOldAPI)
String getUserName ()
RestDistributionQueue setUserName (String userName)
```

```

String getPassword ()
RestDistributionQueue setPassword (String password)
boolean isBatch ()
RestDistributionQueue setBatch (boolean batch)
boolean isAsync ()
RestDistributionQueue setAsync (boolean async)
int getPeriod ()
RestDistributionQueue setPeriod (int period)
Queue create (String dest) throws ConnectException, AdminException
Queue create (int serverId, String dest) throws ConnectException, AdminException
Queue create (int serverId, String name, String dest) throws ConnectException, AdminException
Queue create (int serverId, String name, String dest, Properties props) throws ConnectException, AdminException

```

Static Public Attributes

static final String RESTDistribution = "com.scalagent.joram.mom.dest.rest.RESTDistribution"

Detailed Description

The **RestDistributionQueue** class allows administrators to create REST distribution queues (REST bridge out).

The REST bridge destinations rely on a particular Joram service which purpose is to maintain valid connections with the foreign REST Joram servers.

The valid properties define by this destination are:

- rest.host: hostname or IP address of remote JMS REST server, by default "localhost".
- rest.port: listening port of remote JMS REST server, by default 8989.
- rest.useOldAPI: if true, use the old Rest/JMS API with authentication through query parameters.
- rest.user: user name needed to connect to remote JMS REST server, by default "anonymous".
- rest.pass: password needed to connect to remote JMS REST server, by default "anonymous".
- distribution.async: default true.
- distribution.batch: default true.
- rest.idletimeout: normally each remote resource need to be explicitly closed, this parameter allows to set the idle time in seconds in which the remote context will be closed if idle. If less than or equal to 0, the context never closes. By default 60 seconds.
- period: default 1.000 (1 second).
- jms.destination: the name of remote JMS destination.

Member Function Documentation

String org.objectweb.joram.client.jms.admin.RestDistributionQueue.getHost ()

Returns:

the hostName

RestDistributionQueue

org.objectweb.joram.client.jms.admin.RestDistributionQueue.setHostName (String *hostName*)

Parameters:

<i>host</i>	the hostname or IP address of the remote Rest / JMS service
-------------	---

int org.objectweb.joram.client.jms.admin.RestDistributionQueue.getPort ()

Returns:

the port

RestDistributionQueue org.objectweb.joram.client.jms.admin.RestDistributionQueue.setPort (int port)

Parameters:

<i>port</i>	the listening port of the remote Rest / JMS service
-------------	---

boolean org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.isUseOldAPI()

Returns:

value of the option useOldAPI.

RestAcquisitionQueue

org.objectweb.joram.client.jms.admin.RestAcquisitionQueue.setUseOldAPI (boolean useOldAPI)

Parameters:

<i>useOldAPI</i>	the value of the option useOldAPI.
------------------	------------------------------------

String org.objectweb.joram.client.jms.admin.RestDistributionQueue.getUserName ()

Returns:

the userName

RestDistributionQueue

org.objectweb.joram.client.jms.admin.RestDistributionQueue.setUserName (String userName)

Parameters:

<i>userName</i>	the userName to set
-----------------	---------------------

String org.objectweb.joram.client.jms.admin.RestDistributionQueue.getPassword ()

Returns:

the password

RestDistributionQueue

org.objectweb.joram.client.jms.admin.RestDistributionQueue.setPassword (String password)

Parameters:

<i>password</i>	the password to set
-----------------	---------------------

boolean org.objectweb.joram.client.jms.admin.RestDistributionQueue.isBatch ()

Gets the batch parameter.

Returns:

the batch parameter

RestDistributionQueue org.objectweb.joram.client.jms.admin.RestDistributionQueue.setBatch (boolean batch)

Sets the batch parameter.

Parameters:

<i>batch</i>	the batch parameter to set
--------------	----------------------------

boolean org.objectweb.joram.client.jms.admin.RestDistributionQueue.isAsync ()

Gets the async parameter.

Returns:

the async parameter

RestDistributionQueue org.objectweb.joram.client.jms.admin.RestDistributionQueue.setAsync (boolean *async*)

Sets the async parameter.

Parameters:

<i>async</i>	the async parameter to set
--------------	----------------------------

int org.objectweb.joram.client.jms.admin.RestDistributionQueue.getPeriod ()**Returns:**

the period

RestDistributionQueue org.objectweb.joram.client.jms.admin.RestDistributionQueue.setPeriod (int *period*)**Parameters:**

<i>period</i>	the period to set
---------------	-------------------

Queue org.objectweb.joram.client.jms.admin.RestDistributionQueue.create (String *dest*) throws ConnectException, AdminException

Administration method creating and deploying a REST distribution queue on the local server.

The request fails if the destination deployment fails server side.

Be careful this method use the static **AdminModule** connection.

Parameters:

<i>dest</i>	The name of the foreign destination.
-------------	--------------------------------------

Returns:

the created bridge destination.

Exceptions:

<i>ConnectException</i>	If the administration connection is closed or broken.
-------------------------	---

<i>AdminException</i>	If the request fails.
-----------------------	-----------------------

See also:

`create(int, String, String, Properties)`

Queue org.objectweb.joram.client.jms.admin.RestDistributionQueue.create (int *serverId*, String *dest*) throws ConnectException, AdminException

Administration method creating and deploying a REST distribution queue on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static **AdminModule** connection.

Parameters:

<i>serverId</i>	The identifier of the server where deploying the queue.
<i>dest</i>	The name of the foreign destination.

Returns:

the created bridge destination.

Exceptions:

<i>ConnectException</i>	If the administration connection is closed or broken.
<i>AdminException</i>	If the request fails.

See also:

`create(int, String, String, Properties)`

Queue `org.objectweb.joram.client.jms.admin.RestDistributionQueue.create (int serverId, String name, String dest) throws ConnectException, AdminException`

Administration method creating and deploying a REST distribution queue on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static **AdminModule** connection.

Parameters:

<i>serverId</i>	The identifier of the server where deploying the queue.
<i>name</i>	The name of the created queue.
<i>dest</i>	The name of the foreign destination.

Returns:

the created bridge destination.

Exceptions:

<i>ConnectException</i>	If the administration connection is closed or broken.
<i>AdminException</i>	If the request fails.

See also:

`create(int, String, String, Properties)`

Queue `org.objectweb.joram.client.jms.admin.RestDistributionQueue.create (int serverId, String name, String dest, Properties props) throws ConnectException, AdminException`

Administration method creating and deploying a REST distribution queue on a given server.

In addition to properties used to configure distribution queues a set of specific properties allows to configure Rest/JMS distribution destination::

- rest.host: hostname or IP address of remote JMS REST server, by default "localhost".
- rest.port: listening port of remote JMS REST server, by default 8989.
- rest.user: user name needed to connect to remote JMS REST server, by default "anonymous".
- rest.pass: password needed to connect to remote JMS REST server, by default "anonymous".
- distribution.async: default true.
- distribution.batch: default true.
- rest.idletimeout: normally each remote resource need to be explicitly closed, this parameter allows to set the idle time in seconds in which the remote context will be closed if idle. If less than or equal to 0, the context never closes. By default 60 seconds.
- period: default 1.000 (1 second).
- jms.destination: the name of remote JMS destination.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static **AdminModule** connection.

Parameters:

<i>serverId</i>	The identifier of the server where deploying the queue.
<i>name</i>	The name of the created queue.
<i>dest</i>	The name of the foreign destination.
<i>props</i>	A Properties object containing all needed parameters.

Returns:

the created bridge destination.

Exceptions:

<i>ConnectException</i>	If the administration connection is closed or broken.
<i>AdminException</i>	If the request fails.

Member Data Documentation

```
final String org.objectweb.joram.client.jms.admin.RestDistributionQueue.RESTDistribution = "com.scalagent.joram.mom.dest.rest.RESTDistribution" [static]
```

Class name of handler allowing to distribute messages to a foreign REST provider.

6.7. AMQPAcquisitionQueue

Class `org.objectweb.joram.client.jms.admin.AMQPAcquisitionQueue`

Static Public Member Functions

```
static Queue create (String dest) throws ConnectException, AdminException
static Queue create (int serverId, String dest) throws ConnectException, AdminException
static Queue create (int serverId, String name, String dest) throws ConnectException, AdminException
static Queue create (int serverId, String name, String dest, Properties props) throws ConnectException, AdminException
```

Static Public Attributes

```
static final String AMQPAcquisition = "org.objectweb.joram.mom.dest.amqp.AmqpAcquisition"
```

Detailed Description

The `AMQPAcquisitionQueue` class allows administrators to create AMQP acquisition queues (AMQP bridge in). The AMQP bridge destinations rely on a particular Joram service which purpose is to maintain valid connections with the foreign AMQP servers.

Member Function Documentation

```
static Queue AMQPAcquisitionQueue.create (String dest) throws ConnectException, AdminException [static]
```

Administration method creating and deploying a AMQP acquisition queue on the local server.
The request fails if the destination deployment fails server side.
Be careful this method use the static AdminModule connection.

Parameters:

<i>dest</i>	The name of the foreign destination.
-------------	--------------------------------------

Returns:

the created bridge destination.

Exceptions:

<i>ConnectException</i>	If the administration connection is closed or broken.
<i>AdminException</i>	If the request fails.

static Queue AMQPACquisitionQueue.create (int serverId, String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a AMQP acquisition queue on a given server.
The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
dest	The name of the foreign destination.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Queue AMQPACquisitionQueue.create (int serverId, String name, String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a AMQP acquisition queue on a given server.
The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.
dest	The name of the foreign destination.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Queue AMQPACquisitionQueue.create (int serverId, String name, String dest, Properties props) throws ConnectException, AdminException [static]

Administration method creating and deploying a AMQP acquisition queue on a given server.

A set of properties is used to configure the distribution destination:

- period – .
- acquisition.period - The period between two acquisitions, default is 0 (no periodic acquisition).
- persistent - Tells if produced messages will be persistent, default is true (JMS default).
- expiration - Tells the life expectancy of produced messages, default is 0 (JMS default time to live).
- priority - Tells the JMS priority of produced messages, default is 4 (JMS default).
- acquisition.max_msg - The maximum number of messages between the last message acquired by the handler and the message correctly handled by the acquisition destination, default is 20. When the number of messages waiting to be handled is greater the acquisition handler is temporarily stopped. A value lesser or equal to 0 disables the mechanism.
- acquisition.min_msg - The minimum number of message to restart the acquisition, default is 10.
- acquisition.max_pnd - The maximum number of pending messages on the acquisition destination, default is 20. When the number of waiting messages is greater the acquisition handler is temporarily stopped. A value lesser or equal to 0 disables the mechanism.
- acquisition.min_pnd - The minimum number of pending messages to restart the acquisition, default is 10.
- amqp.ConnectionUpdatePeriod - Period between two update phases allowing the discovering of new Connections. Default value is 5000.

- amqp.Queue.DeclarePassive – If true declare a queue passively; i.e., check if it exists but do not create it. If false the queue is created if it does not exist. Default value is true.
- amqp.Queue.DeclareExclusive – If true we are declaring an exclusive queue (restricted to this connection). Default value is false.
- amqp.Queue.DeclareDurable – If true we are declaring a durable queue (the queue will survive a server restart). Default value is true.
- amqp.Queue.DeclareAutoDelete – If true we are declaring an “autodelete” queue (server will delete it when no longer in use). Default value is false.
- amqp.Routing - This property allows to filter the connections used to acquire messages.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.
dest	The name of the foreign destination.
props	A Properties object containing all needed parameters.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

Member Data Documentation

final String AMQPAcquisitionQueue.AMQPAcquisition = "org.objectweb.joram.mom.dest.amqp.AmqpAcquisition" [static]

Class name of handler allowing to acquire messages to a foreign AMQP provider.

6.8. AMQPAcquisitionTopic

Class org.objectweb.joram.client.jms.admin.AMQPAcquisitionTopic

Static Public Member Functions

static Topic create (String dest) throws ConnectException, AdminException
 static Topic create (int serverId, String dest) throws ConnectException, AdminException
 static Topic create (int serverId, String name, String dest) throws ConnectException, AdminException
 static Topic create (int serverId, String name, String dest, Properties props) throws ConnectException, AdminException

Static Public Attributes

static final String AMQPAcquisition = "org.objectweb.joram.mom.dest.amqp.AmqpAcquisition"

Detailed Description

The AMQPAcquisitionTopic class allows administrators to create AMQP acquisition topics (AMQP bridge in). The AMQP bridge destinations rely on a particular Joram service which purpose is to maintain valid connections with the foreign AMQP servers.

Member Function Documentation

static Topic AMQPAcquisitionTopic.create (String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a AMQP acquisition topic on the local server.

The request fails if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

dest	The name of the foreign destination.
------	--------------------------------------

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Topic AMQPAcquisitionTopic.create (int serverId, String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a AMQP acquisition topic on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
dest	The name of the foreign destination.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Topic AMQPAcquisitionTopic.create (int serverId, String name, String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a AMQP acquisition topic on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the created topic.
dest	The name of the foreign destination.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Topic AMQPAcquisitionTopic.create (int serverId, String name, String dest, Properties props) throws ConnectException, AdminException [static]

Administration method creating and deploying a AMQP acquisition topic on a given server.

A set of properties is used to configure the distribution destination:

- period – .
- acquisition.period - The period between two acquisitions, default is 0 (no periodic acquisition).
- persistent - Tells if produced messages will be persistent, default is true (JMS default).
- expiration - Tells the life expectancy of produced messages, default is 0 (JMS default time to live).
- priority - Tells the JMS priority of produced messages, default is 4 (JMS default).
- amqp.ConnectionUpdatePeriod - Period between two update phases allowing the discovering of new Connections. Default value is 5000.
- amqp.Queue.DeclarePassive – If true declare a queue passively; i.e., check if it exists but do not create it. If false the queue is created if it does not exist. Default value is true.
- amqp.Queue.DeclareExclusive – If true we are declaring an exclusive queue (restricted to this connection). Default value is false.
- amqp.Queue.DeclareDurable – If true we are declaring a durable queue (the queue will survive a server restart). Default value is true.
- amqp.Queue.DeclareAutoDelete – If true we are declaring an “autodelete” queue (server will delete it when no longer in use). Default value is false.
- amqp.Routing - This property allows to filter the connections used to acquire messages.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the created topic.
dest	The name of the foreign destination.
props	A Properties object containing all needed parameters.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

Member Data Documentation

final String AMQPAcquisitionTopic.AMQPAcquisition "org.objectweb.joram.mom.dest.amqp.AmqpAcquisition" [static] =

Class name of handler allowing to acquire messages to a foreign AMQP provider.

6.9. AMQPDistributionQueue

Class org.objectweb.joram.client.jms.admin.AMQPDistributionQueue

Static Public Member Functions

static Queue create (String dest) throws ConnectException, AdminException
 static Queue create (int serverId, String dest) throws ConnectException, AdminException
 static Queue create (int serverId, String name, String dest) throws ConnectException, AdminException
 static Queue create (int serverId, String name, String dest, Properties props) throws ConnectException, AdminException

Static Public Attributes

static final String AMQPDistribution = "org.objectweb.joram.mom.dest.amqp.AmqpDistribution"

Detailed Description

The AMQPDistributionQueue class allows administrators to create AMQP distribution queues (AMQP bridge out). The AMQP bridge destinations rely on a particular Joram service which purpose is to maintain valid connections with the foreign XMQ servers.

Member Function Documentation

static Queue AMQPDistributionQueue.create (String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a AMQP distribution queue on the local server.
The request fails if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

dest	The name of the foreign destination.
------	--------------------------------------

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Queue AMQPDistributionQueue.create (int serverId, String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a AMQP distribution queue on a given server.
The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
dest	The name of the foreign destination.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Queue AMQPDistributionQueue.create (int serverId, String name, String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a AMQP distribution queue on a given server.
The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.

dest	The name of the foreign destination.
------	--------------------------------------

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Queue AMQPDistributionQueue.create (int serverId, String name, String dest, Properties props) throws ConnectException, AdminException [static]

Administration method creating and deploying a AMQP distribution queue on a given server.

A set of properties is used to configure the distribution destination:

- period – Tells the time to wait before another distribution attempt. Default is 0, which means there won't be other attempts.
- distribution.batch – If set to true, the destination will try to distribute each time every waiting message, regardless of distribution errors. This can lead to the loss of message ordering, but will prevent a blocking message from blocking every following message. When set to false, the distribution process will stop on the first error. Default is false.
- distribution.async - If set to true, the messages are asynchronously forwarded through a daemon.
- amqp.ConnectionUpdatePeriod - Period between two update phases allowing the discovering of new Connections. Default value is 5000.
- amqp.Queue.DeclarePassive – If true declare a queue passively; i.e., check if it exists but do not create it. If false the queue is created if it does not exist. Default value is true.
- amqp.Queue.DeclareExclusive – If true we are declaring an exclusive queue (restricted to this connection). Default value is false.
- amqp.Queue.DeclareDurable – If true we are declaring a durable queue (the queue will survive a server restart). Default value is true.
- amqp.Queue.DeclareAutoDelete – If true we are declaring an “autodelete” queue (server will delete it when no longer in use). Default value is false.
- amqp.Routing - This property allows to filter the connections used to forward the messages. It can be set either globally at creation or specifically for a message at sending.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.
dest	The name of the foreign destination.
props	A Properties object containing all needed parameters.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

Member Data Documentation**final String AMQPDistributionQueue.AMQPDistribution = "org.objectweb.joram.mom.dest.amqp.AmqpDistribution" [static]**

Class name of handler allowing to distribute messages to a foreign AMQP provider.

6.10. AMQP Distribution Topic

Class org.objectweb.joram.client.jms.admin.AMQPDistributionTopic

Static Public Member Functions

```
static Topic create (String dest) throws ConnectException, AdminException
static Topic create (int serverId, String dest) throws ConnectException, AdminException
static Topic create (int serverId, String name, String dest) throws ConnectException, AdminException
static Topic create (int serverId, String name, String dest, Properties props) throws ConnectException, AdminException
```

Static Public Attributes

```
static final String AMQPDistribution = "org.objectweb.joram.mom.dest.amqp.AmqpDistribution"
```

Detailed Description

The AMQPDistributionTopic class allows administrators to create AMQP distribution topics (AMQP bridge out). The AMQP bridge destinations rely on a particular Joram service which purpose is to maintain valid connections with the foreign XMQ servers.

Member Function Documentation

static Topic AMQPDistributionTopic.create (String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a AMQP distribution topic on the local server.
The request fails if the destination deployment fails server side.
Be careful this method use the static AdminModule connection.

Parameters:

dest	The name of the foreign destination.
------	--------------------------------------

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Topic AMQPDistributionTopic.create (int serverId, String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a AMQP distribution topic on a given server.
The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.
Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
dest	The name of the foreign destination.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Topic AMQPDistributionTopic.create (int serverId, String name, String dest) throws ConnectException, AdminException [static]

Administration method creating and deploying a AMQP distribution topic on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the created topic.
dest	The name of the foreign destination.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Topic AMQPDistributionTopic.create (int serverId, String name, String dest, Properties props) throws ConnectException, AdminException [static]

Administration method creating and deploying a AMQP distribution topic on a given server.

A set of properties is used to configure the distribution destination:

- period – Tells the time to wait before another distribution attempt. Default is 0, which means there won't be other attempts.
- distribution.batch – If set to true, the destination will try to distribute each time every waiting message, regardless of distribution errors. This can lead to the loss of message ordering, but will prevent a blocking message from blocking every following message. When set to false, the distribution process will stop on the first error. Default is false.
- distribution.async - If set to true, the messages are asynchronously forwarded through a daemon.
- amqp.ConnectionUpdatePeriod - Period between two update phases allowing the discovering of new Connections. Default value is 5000.
- amqp.Queue.DeclarePassive – If true declare a queue passively; i.e., check if it exists but do not create it. If false the queue is created if it does not exist. Default value is true.
- amqp.Queue.DeclareExclusive – If true we are declaring an exclusive queue (restricted to this connection). Default value is false.
- amqp.Queue.DeclareDurable – If true we are declaring a durable queue (the queue will survive a server restart). Default value is true.
- amqp.Queue.DeclareAutoDelete – If true we are declaring an “autodelete” queue (server will delete it when no longer in use). Default value is false.
- amqp.Routing - This property allows to filter the connections used to forward the messages. It can be set either globally at creation or specifically for a message at sending.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the created topic.
dest	The name of the foreign destination.
props	A Properties object containing all needed parameters.

Returns:

the created bridge destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

Member Data Documentation

final String AMQPDistributionTopic.AMQPDistribution = "org.objectweb.joram.mom.dest.amqp.AmqpDistribution" [static]

Class name of handler allowing to distribute messages to a foreign AMQP provider.

6.11. CollectorQueue

Class org.objectweb.joram.client.jms.admin.CollectorQueue

Static Public Member Functions

```
static Queue create (String url) throws ConnectException, AdminException
static Queue create (int serverId, String url) throws ConnectException, AdminException
static Queue create (int serverId, String name, String url) throws ConnectException, AdminException
static Queue create (int serverId, String name, String url, Properties props) throws ConnectException, AdminException
```

Static Public Attributes

static final String URLAcquisition = "com.scalagent.joram.mom.dest.collector.URLAcquisition"

Detailed Description

The CollectorQueue class allows administrators to create JMS collector queues.

Collector queues are special destinations usable to collect a document from a specified URL. They can be used to periodically import a file from an URL to a Joram's message and store it in this queue.

Member Function Documentation

static Queue CollectorQueue.create (String url) throws ConnectException, AdminException [static]

Administration method creating and deploying an URL acquisition queue on the local server.

The request fails if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

url	the URL locating the element to collect.
-----	--

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Queue CollectorQueue.create (int serverId, String url) throws ConnectException, AdminException [static]

Administration method creating and deploying an URL acquisition queue on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
url	the URL locating the element to collect.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Queue CollectorQueue.create (int serverId, String name, String url) throws ConnectException, AdminException [static]

Administration method creating and deploying an URL acquisition queue on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.
url	the URL locating the element to collect.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Queue CollectorQueue.create (int serverId, String name, String url, Properties props) throws ConnectException, AdminException [static]

Administration method creating and deploying an URL acquisition queue on a given server.

A set of properties is used to configure the distribution destination:

- period – .
- acquisition.period - The period between two acquisitions, default is 0 (no periodic acquisition). If this last case the acquisition must be triggered by an incoming message, the destination properties can then be overloaded by the message ones.
- persistent - Tells if produced messages will be persistent, default is true (JMS default).
- expiration - Tells the life expectancy of produced messages, default is 0 (JMS default time to live).
- priority - Tells the JMS priority of produced messages, default is 4 (JMS default).
- collector.url - locates the element that will be collected.
- collector.type - indicates the type of the generated message. Default is Message.BYTES.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.
url	the URL locating the element to collect.

props	A Properties object containing all needed parameters.
-------	---

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

Member Data Documentation

final String CollectorQueue.URLAcquisition = "com.scalagent.joram.mom.dest.collector.URLAcquisition" [static]

Class name of handler allowing to acquire messages from a specified URL.

6.12. CollectorTopic

Class org.objectweb.joram.client.jms.admin.CollectorTopic

Static Public Member Functions

```
static Topic create (String url) throws ConnectException, AdminException
static Topic create (int serverId, String url) throws ConnectException, AdminException
static Topic create (int serverId, String name, String url) throws ConnectException, AdminException
static Topic create (int serverId, String name, String url, Properties props) throws ConnectException, AdminException
```

Static Public Attributes

static final String URLAcquisition = "com.scalagent.joram.mom.dest.collector.URLAcquisition"

Detailed Description

The CollectorTopic class allows administrators to create JMS collector topics (JMS bridge in). Collector topics are special destinations usable to collect a document from a specified URL. They can be used to periodically import a file from an URL to a Joram's message and forward to each subscribers.

Member Function Documentation

static Topic CollectorTopic.create (String url) throws ConnectException, AdminException [static]

Administration method creating and deploying an URL acquisition topic on the local server.

The request fails if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

url	the URL locating the element to collect.
-----	--

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Topic CollectorTopic.create (int serverId, String url) throws ConnectException, AdminException [static]

Administration method creating and deploying an URL acquisition topic on a given server.
The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
url	the URL locating the element to collect.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Topic CollectorTopic.create (int serverId, String name, String url) throws ConnectException, AdminException [static]

Administration method creating and deploying an URL acquisition topic on a given server.
The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the created topic.
url	the URL locating the element to collect.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Topic CollectorTopic.create (int serverId, String name, String url, Properties props) throws ConnectException, AdminException [static]

Administration method creating and deploying an URL acquisition topic on a given server.

A set of properties is used to configure the distribution destination:

- period – .
- acquisition.period - The period between two acquisitions, default is 0 (no periodic acquisition). If this last case the acquisition must be triggered by an incoming message, the destination properties can then be overloaded by the message ones.
- persistent - Tells if produced messages will be persistent, default is true (JMS default).
- expiration - Tells the life expectancy of produced messages, default is 0 (JMS default time to live).
- priority - Tells the JMS priority of produced messages, default is 4 (JMS default).
- collector.url - locates the element that will be collected.
- collector.type - indicates the type of the generated message. Default is Message.BYTES.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the created topic.
url	the URL locating the element to collect.

props	A Properties object containing all needed parameters.
-------	---

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

Member Data Documentation

```
final String org.objectweb.joram.client.jms.admin.CollectorTopic.URLAcquisition =  
"com.scalagent.joram.mom.dest.collector.URLAcquisition" [static]
```

Class name of handler allowing to acquire messages from a specified URL.

6.13. FtpQueue

Class org.objectweb.joram.client.jms.admin.FtpQueue

Static Public Member Functions

```
static Queue create (String name) throws ConnectException, AdminException  
static Queue create (int serverId, String name) throws ConnectException, AdminException  
static Queue create (int serverId, String name, Properties props) throws ConnectException, AdminException
```

Static Public Attributes

```
static final String DefaultFTPImpl = "com.scalagent.joram.mom.dest.ftp.TransferImplRef"  
static final String JFTPImpl = "com.scalagent.joram.mom.dest.ftp.TransferImplJftp"
```

Detailed Description

The FtpQueue class allows administrators to create FTP queues.

A FTP queue is special destinations allowing to transfer file by FTP. It wraps the FTP transfer of a file through a message exchange. The sender starts the transfer by sending a JMS message, and the receiver is notified of the transfer completion by a JMS message.

Member Function Documentation

```
static Queue org.objectweb.joram.client.jms.admin.FtpQueue.create (String name) throws  
ConnectException, AdminException [static]
```

Administration method creating and deploying a FTP queue on the local server.

The request fails if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

name	The name of the created queue.
------	--------------------------------

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
------------------	---

AdminException	If the request fails.
----------------	-----------------------

static Queue org.objectweb.joram.client.jms.admin.FtpQueue.create (int serverId, String name) throws ConnectException, AdminException [static]

Administration method creating and deploying a FTP queue on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Queue org.objectweb.joram.client.jms.admin.FtpQueue.create (int serverId, String name, Properties props) throws ConnectException, AdminException [static]

Administration method creating and deploying a FTP queue on a given server.

A set of properties is used to configure the FTP destination:

- user - the user name for the FTP.
 - pass - the user password for FTP.
 - path - the local directory to store the file transferred. Default is the running directory (path=null).
 - ftpImplName: The implementation of the FTP transfer, we provide two implementations:
 - the default one based on JDK URL:.scalagent.joram.mom.dest.ftp.TransferImplRef
 - the second is based on JFTP:.scalagent.joram.mom.dest.ftp.TransferImplJftp.
- 1 The user and pass options are optional, because this information can be set through an URL by the sender as a message property like this:
- msg.setStringProperty("url", "ftp://user:pass@host/file?type=i");

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.
props	A Properties object containing all needed parameters.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

Member Data Documentation

final String org.objectweb.joram.client.jms.admin.FtpQueue.DefaultFTPImpl = "com.scalagent.joram.mom.dest.ftp.TransferImplRef" [static]

Class name of default handler allowing to transfer file through FTP.

```
final String org.objectweb.joram.client.jms.admin.FtpQueue.JFTPImpl
"com.scalagent.joram.mom.dest.ftp.TransferImplJftp" [static]
```

Class name of handler allowing to transfer file using JFTP.

6.14. MailAcquisitionQueue

Class org.objectweb.joram.client.jms.admin.MailAcquisitionQueue

Static Public Member Functions

```
static Queue create () throws ConnectException, AdminException
static Queue create (int serverId) throws ConnectException, AdminException
static Queue create (int serverId, String name) throws ConnectException, AdminException
static Queue create (int serverId, String name, Properties props) throws ConnectException, AdminException
```

Static Public Attributes

```
static final String MailAcquisition = "com.scalagent.joram.mom.dest.mail.MailAcquisition"
```

Detailed Description

The `MailAcquisitionQueue` class allows administrators to create Mail acquisition queues. Using an e-mail account, mail acquisition destinations allow you to import emails from this external account using POP and turn them into Joram's JMS messages.

Member Function Documentation

static Queue MailAcquisitionQueue.create () throws ConnectException, AdminException [static]

Administration method creating and deploying a mail acquisition queue on the local server.

The request fails if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Queue MailAcquisitionQueue.create (int serverId) throws ConnectException, AdminException [static]

Administration method creating and deploying a mail acquisition queue on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
----------	---

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Queue MailAcquisitionQueue.create (int serverId, String name) throws ConnectException, AdminException [static]

Administration method creating and deploying a mail acquisition queue on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Queue MailAcquisitionQueue.create (int serverId, String name, Properties props) throws ConnectException, AdminException [static]

Administration method creating and deploying a mail acquisition queue on a given server.

A set of properties is used to configure the acquisition destination:

- period – .
- acquisition.period - The period between two acquisitions, default is 0 (no periodic acquisition).
- persistent - Tells if produced messages will be persistent, default is true (JMS default).
- expiration - Tells the life expectancy of produced messages, default is 0 (JMS default time to live).
- priority - Tells the JMS priority of produced messages, default is 4 (JMS default).
- popServer - the DNS name or IP address of the POP server.
- popUser - the login name for the email account.
- popPassword - the password for the email account.
- expunge - allows to remove or not email on the server.
- The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.
props	A Properties object containing all needed parameters.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

Member Data Documentation

final String MailAcquisitionQueue.MailAcquisition "com.scalagent.joram.mom.dest.mail.MailAcquisition" [static]

Class name of handler allowing to acquire messages from a POP mail provider.

This handler is used by default to create MailAcquisitionQueue, the acquisition.className property allows to declare an alternate handler using different protocol (IMAP for example).

6.15. MailAcquisitionTopic

Class org.objectweb.joram.client.jms.admin.MailAcquisitionTopic

Static Public Member Functions

```
static Topic create () throws ConnectException, AdminException
static Topic create (int serverId) throws ConnectException, AdminException
static Topic create (int serverId, String name) throws ConnectException, AdminException
static Topic create (int serverId, String name, Properties props) throws ConnectException, AdminException
```

Static Public Attributes

```
static final String MailAcquisition = "com.scalagent.joram.mom.dest.mail.MailAcquisition"
```

Detailed Description

The `MailAcquisitionTopic` class allows administrators to create Mail acquisition topics. Using an e-mail account, mail acquisition destinations allow you to import emails from this external account using POP and turn them into Joram's JMS messages.

Member Function Documentation

static Topic MailAcquisitionTopic.create () throws ConnectException, AdminException [static]

Administration method creating and deploying a mail acquisition topic on the local server.

The request fails if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Topic MailAcquisitionTopic.create (int serverId) throws ConnectException, AdminException [static]

Administration method creating and deploying a mail acquisition topic on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
----------	---

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Topic MailAcquisitionTopic.create (int serverId, String name) throws ConnectException, AdminException [static]

Administration method creating and deploying a mail acquisition topic on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Topic org.objectweb.joram.client.jms.admin.MailAcquisitionTopic.create (int serverId, String name, Properties props) throws ConnectException, AdminException [static]

Administration method creating and deploying a mail acquisition topic on a given server.

A set of properties is used to configure the acquisition destination:

- period – .
- acquisition.period - The period between two acquisitions, default is 0 (no periodic acquisition).
- persistent - Tells if produced messages will be persistent, default is true (JMS default).
- expiration - Tells the life expectancy of produced messages, default is 0 (JMS default time to live).
- priority - Tells the JMS priority of produced messages, default is 4 (JMS default).
- popServer - the DNS name or IP address of the POP server.
- popUser - the login name for the email account.
- popPassword - the password for the email account.
- expunge - allows to remove or not email on the server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.
props	A Properties object containing all needed parameters.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

Member Data Documentation

final String org.objectweb.joram.client.jms.admin.MailAcquisitionTopic.MailAcquisition = "com.scalagent.joram.mom.dest.mail.MailAcquisition" [static]

Class name of handler allowing to acquire messages from a POP mail provider.

This handler is used by default to create MailAcquisitionTopic, the acquisition.className property allows to declare an alternate handler using different protocol (IMAP for example).

6.16. MailDistributionQueue

Class org.objectweb.joram.client.jms.admin.MailDistributionQueue

Static Public Member Functions

```
static Queue create () throws ConnectException, AdminException
static Queue create (int serverId) throws ConnectException, AdminException
static Queue create (int serverId, String name) throws ConnectException, AdminException
static Queue create (int serverId, String name, Properties props) throws ConnectException, AdminException
```

Static Public Attributes

```
static final String MailDistribution = "com.scalagent.joram.mom.dest.mail.MailDistribution"
```

Detailed Description

The MailDistributionQueue class allows administrators to create Mail distribution queues. Using an e-mail account, mail destinations allow you to forward Joram's messages to an external email account using SMTP.

Member Function Documentation

static Queue MailDistributionQueue.create () throws ConnectException, AdminException [static]

Administration method creating and deploying a Mail distribution queue on the local server. The request fails if the destination deployment fails server side. Be careful this method use the static AdminModule connection.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Queue MailDistributionQueue.create (int serverId) throws ConnectException, AdminException [static]

Administration method creating and deploying a Mail distribution queue on a given server. The request fails if the target server does not belong to the platform, or if the destination deployment fails server side. Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
----------	---

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Queue org.objectweb.joram.client.jms.admin.MailDistributionQueue.create (int serverId, String name) throws ConnectException, AdminException [static]

Administration method creating and deploying a Mail distribution queue on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Queue org.objectweb.joram.client.jms.admin.MailDistributionQueue.create (int serverId, String name, Properties props) throws ConnectException, AdminException [static]

Administration method creating and deploying a Mail distribution queue on a given server.

A set of properties is used to configure the distribution destination:

- period – Tells the time to wait before another distribution attempt. Default is 0, which means there won't be other attempts.
- distribution.batch – If set to true, the destination will try to distribute each time every waiting message, regardless of distribution errors. This can lead to the loss of message ordering, but will prevent a blocking message from blocking every following message. When set to false, the distribution process will stop on the first error. Default is false.
- distribution.async - If set to true, the messages are asynchronously forwarded through a daemon.
- smtpServer - the DNS name or IP address of the SMTP server.
- from - the email address of the sender.
- to, cc, bcc - a comma separated list of recipients.
- subject - the subject of outgoing message.
- selector - additionally a selector can be added to filter the forwarded messages.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.
props	A Properties object containing all needed parameters.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

Member Data Documentation

final String org.objectweb.joram.client.jms.admin.MailDistributionQueue.MailDistribution = "com.scalagent.joram.mom.dest.mail.MailDistribution" [static]

Class name of handler allowing to distribute messages to a SMTP mail server.

This handler is used by default to create MailDistributionQueue, the distribution.className property allows to declare an alternate handler using different protocol or implementation.

6.17. MailDistributionTopic

Class org.objectweb.joram.client.jms.admin.MailDistributionTopic

Static Public Member Functions

```
static Topic create () throws ConnectException, AdminException
static Topic create (int serverId) throws ConnectException, AdminException
static Topic create (int serverId, String name) throws ConnectException, AdminException
static Topic create (int serverId, String name, Properties props) throws ConnectException, AdminException
```

Static Public Attributes

```
static final String MailDistribution = "com.scalagent.joram.mom.dest.mail.MailDistribution"
```

Detailed Description

The MailDistributionTopic class allows administrators to create Mail distribution topics. Using an e-mail account, mail destinations allow you to forward Joram's messages to an external email account using SMTP.

Member Function Documentation

static Topic MailDistributionTopic.create () throws ConnectException, AdminException [static]

Administration method creating and deploying a Mail distribution topic on the local server.

The request fails if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Topic MailDistributionTopic.create (int serverId) throws ConnectException, AdminException [static]

Administration method creating and deploying a Mail distribution topic on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
----------	---

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Topic MailDistributionTopic.create (int serverId, String name) throws ConnectException, AdminException [static]

Administration method creating and deploying a Mail distribution topic on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the created topic.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Topic MailDistributionTopic.create (int serverId, String name, Properties props) throws ConnectException, AdminException [static]

Administration method creating and deploying a Mail distribution topic on a given server.

A set of properties is used to configure the distribution destination:

- period – Tells the time to wait before another distribution attempt. Default is 0, which means there won't be other attempts.
- distribution.batch – If set to true, the destination will try to distribute each time every waiting message, regardless of distribution errors. This can lead to the loss of message ordering, but will prevent a blocking message from blocking every following message. When set to false, the distribution process will stop on the first error. Default is false.
- distribution.async - If set to true, the messages are asynchronously forwarded through a daemon.
- smtpServer - the DNS name or IP address of the SMTP server.
- from - the email address of the sender.
- to, cc, bcc - a comma separated list of recipients.
- subject - the subject of outgoing message.
- selector - additionally a selector can be added to filter the forwarded messages.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the created topic.
props	A Properties object containing all needed parameters.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

Member Data Documentation

final String MailDistributionTopic.MailDistribution = "com.scalagent.joram.mom.dest.mail.MailDistribution" [static]

Class name of handler allowing to distribute messages to a SMTP mail server.

This handler is used by default to create MailDistributionTopic, the distribution.className property allows to declare an alternate handler using different protocol or implementation.

6.18. MonitoringQueue

Class org.objectweb.joram.client.jms.admin.MonitoringQueue

Static Public Member Functions

```
static Queue create () throws ConnectException, AdminException
static Queue create (int serverId) throws ConnectException, AdminException
static Queue create (int serverId, String name) throws ConnectException, AdminException
static Queue create (int serverId, String name, Properties props) throws ConnectException, AdminException
```

Static Public Attributes

```
static final String JMXAcquisition = "org.objectweb.joram.mom.dest.MonitoringAcquisition"
```

Detailed Description

The MonitoringQueue class allows administrators to create acquisition queue for JMX monitoring data. A monitoring acquisition destination is an acquisition destination configured to transform JMX monitoring information into JMS messages. It works in 2 modes:
If the acquisition.period attribute is set (value greater than 0) the destination periodically scans the selected JMX attributes and generates a message with the value of these attributes.
In the other case, the user must send to the destination a message with the list of JMX attributes to scan and the destination creates a message with these values, or use the last known JMX attributes.
Each message is delivered to an unique consumer.
This queue is based on JMX monitoring so you must enable JMX monitoring to use it.

Member Function Documentation

static Queue MonitoringQueue.create () throws ConnectException, AdminException [static]

Administration method creating and deploying a JMX acquisition queue on the local server.
The request fails if the destination deployment fails server side.
Be careful this method use the static AdminModule connection.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Queue org.objectweb.joram.client.jms.admin.MonitoringQueue.create (int serverId) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMX acquisition queue on a given server.
The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.
Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
----------	---

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
------------------	---

AdminException	If the request fails.
----------------	-----------------------

static Queue MonitoringQueue.create (int serverId, String name) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMX acquisition queue on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Queue MonitoringQueue.create (int serverId, String name, Properties props) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMX acquisition queue on a given server.

A set of properties is used to configure the distribution destination:

- period – .
- acquisition.period - The period between two acquisitions, default is 0 (no periodic acquisition). If this last case the acquisition must be triggered by an incoming message, the destination properties can then be overloaded by the message ones.
- persistent - Tells if produced messages will be persistent, default is true (JMS default).
- expiration - Tells the life expectancy of produced messages, default is 0 (JMS default time to live).
- priority - Tells the JMS priority of produced messages, default is 4 (JMS default).
- Additionally to common acquisition parameters (see above), properties are used to indicate the list of JMX attributes that will be monitored. The property key is the name of the MBean and the value is a comma separated list of attributes to monitor for this MBean. The '*' character is allowed to monitor every parameter of the MBean. Accessing multiple MBeans is possible using wildcard characters, as defined in the javax.management.ObjectName class.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.
props	A Properties object containing all needed parameters.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

Member Data Documentation

final String MonitoringQueue.JMXAcquisition = "org.objectweb.joram.mom.dest.MonitoringAcquisition" [static]

Class name of handler allowing to acquire JMX monitoring data.

6.19. MonitoringTopic

Class org.objectweb.joram.client.jms.admin.MonitoringTopic

Static Public Member Functions

```
static Topic create () throws ConnectException, AdminException
static Topic create (int serverId) throws ConnectException, AdminException
static Topic create (int serverId, String name) throws ConnectException, AdminException
static Topic create (int serverId, String name, Properties props) throws ConnectException, AdminException
```

Static Public Attributes

```
static final String JMXAcquisition = "org.objectweb.joram.mom.dest.MonitoringAcquisition"
```

Detailed Description

The MonitoringTopic class allows administrators to create acquisition topic for JMX monitoring data. A monitoring acquisition destination is an acquisition destination configured to transform JMX monitoring information into JMS messages. It works in 2 modes:
If the acquisition.period attribute is set (value greater than 0) the destination periodically scans the selected JMX attributes and generates a message with the value of these attributes.
In the other case, the user must send to the destination a message with the list of JMX attributes to scan and the destination creates a message with these values, or use the last known JMX attributes.
Each message is delivered to all registered subscribers.
This topic is based on JMX monitoring so you must enable JMX monitoring to use it.

Member Function Documentation

static Topic MonitoringTopic.create () throws ConnectException, AdminException [static]

Administration method creating and deploying a JMX acquisition topic on the local server.
The request fails if the destination deployment fails server side.
Be careful this method use the static AdminModule connection.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Topic MonitoringTopic.create (int serverId) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMX acquisition topic on a given server.
The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.
Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
----------	---

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
------------------	---

AdminException	If the request fails.
----------------	-----------------------

static Topic MonitoringTopic.create (int serverId, String name) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMX acquisition topic on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the created topic.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Topic MonitoringTopic.create (int serverId, String name, Properties props) throws ConnectException, AdminException [static]

Administration method creating and deploying a JMX acquisition topic on a given server.

A set of properties is used to configure the distribution destination:

- period – .
- acquisition.period - The period between two acquisitions, default is 0 (no periodic acquisition). If this last case the acquisition must be triggered by an incoming message, the destination properties can then be overloaded by the message ones.
- persistent - Tells if produced messages will be persistent, default is true (JMS default).
- expiration - Tells the life expectancy of produced messages, default is 0 (JMS default time to live).
- priority - Tells the JMS priority of produced messages, default is 4 (JMS default).
- Additionally to common acquisition parameters (see above), properties are used to indicate the list of JMX attributes that will be monitored. The property key is the name of the MBean and the value is a comma separated list of attributes to monitor for this MBean. The '*' character is allowed to monitor every parameter of the MBean. Accessing multiple MBeans is possible using wildcard characters, as defined in the javax.management.ObjectName class.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the created topic.
props	A Properties object containing all needed parameters.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

Member Data Documentation

final String MonitoringTopic.JMXAcquisition = "org.objectweb.joram.mom.dest.MonitoringAcquisition" [static]

Class name of handler allowing to acquire JMX monitoring data.

6.20. SchedulerQueue

Class org.objectweb.joram.client.jms.admin.SchedulerQueue

Static Public Member Functions

```
static Queue create (String name) throws ConnectException, AdminException
static Queue create (int serverId, String name) throws ConnectException, AdminException
```

Detailed Description

The SchedulerQueue class allows administrators to create scheduled queues. A scheduled queue is a standard JMS queue extended with a timer behavior. When a scheduler queue receives a message with a property called 'scheduleDate' (typed as a long) then the message is not available for delivery before the date specified by the property.

Member Function Documentation

static Queue SchedulerQueue.create (String name) throws ConnectException, AdminException [static]

Administration method creating and deploying a scheduled queue on the local server.

The request fails if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

name	The name of the created queue.
------	--------------------------------

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

static Queue SchedulerQueue.create (int serverId, String name) throws ConnectException, AdminException [static]

Administration method creating and deploying a scheduled queue on a given server.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Be careful this method use the static AdminModule connection.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the created queue.

Returns:

the created destination.

Exceptions:

ConnectException	If the administration connection is closed or broken.
AdminException	If the request fails.

7. Message Interception

There are two different ways to intercept messages:

- Client-side: Configured on the ConnectionFactory the client interceptor allows to intercept each message sent or received through the client connection. The messages are handled in the client's context before the sending or after the receiving.
- Server-side: Such an interceptor can be used to handle messages during their transit in the MOM¹. The messages are handled in the server's secure context, only the authorized users can manage this feature, either when the messages enter or leave the server, or when they are delivered to the destination.

The architecture is designed for flexibility so you can chain as many interceptors objects as you want. Using a chain you can achieve complex patterns and add functionality by composition.

Sample features you can do with an Interceptor are for example:

- Logging, auditing or validating messages
- Alerts,
- Filtering,
- Content enrichment and tagging,
- and many other cases according to your imagination.

7.1. Client-side interceptor

JORAM allows special objects called interceptors to intercept and handle each message entering or exiting the client context through a configured connection.

7.1.1. Writing an interceptor

Writing a client-side interceptor is quite easy, just create a class that implements the `MessageInterceptor`² interface (see below). This interface defines a unique method `handle`, The method parameters are the JMS message and the current session. This method does not return values and must not throw exceptions. By convention, the implementation can modify the original message or the current runtime context.

7.1.2. MessageInterceptor Interface

Interface `org.objectweb.joram.client.jms.MessageInterceptor`

Public Member Functions

`void handle (Message message, Session session)`

Detailed Description

Session level message interceptor interface.

The `MessageInterceptor` should be implemented by any class whose instances are intended to intercept either or both following operations within a JMS Session:

- sending a message
- receiving a message

The interceptor can be attached to a JMS Session through configuration (see `joramAdmin.xml` or `FactoryParameters` class) as IN (when consuming a message) or OUT (when producing a message) interceptor.

¹ either when the messages enter or leave the server, or when they are delivered to the destination.

² Package `org.objectweb.joram.client.jms`

Member Function Documentation

void MessageInterceptor.handle (Message message, Session session)

This method handles a message before proceeding. convention, the implementation can modify the original message or the current runtime context, and return no out value. It also avoids to throw any exception within this method.

Parameters:

message	the message to handle.
session	the current session of the JMS interaction

7.1.3. Configuring a ConnectionFactory with interceptors

The interceptor chain is activated on each message sent or received by the configured session. For a sent message the processing is done before the transfer, for a received message it is done between the receipt of the message client side and the return to the message consumer.

Normally the interceptors chains are configured at the creation of the ConnectionFactory administered object. It can be achieved either through the administration API or the XML scripting langage. The configuration is done on the ConnectionFactory by adding (or removing) individually each interceptor (see [FactoryParameters](#)).

```
public void addInInterceptor(String pInterceptorClassName);
public boolean removeInInterceptor(String pInterceptorClassName);
```

```
public void addOutInterceptor(String pInterceptorClassName);
public boolean removeOutInterceptor(String pInterceptorClassName);
```

In order to configure the interceptors of a destination in a XML script you simply have to add the definition of the interceptor's chains in the ConnectionFactory declaration (see [XML declaration of ConnectionFactory](#)).

For example:

```
<ConnectionFactory name="..." className="...">

...
<inInterceptors>
    <interceptor className="interceptor IN classname #1"/>
    <interceptor className="interceptor IN classname #2"/>
</inInterceptors>

<outInterceptors>
    <interceptor className="interceptor OUT classname #1"/>
</outInterceptors>
...
</ConnectionFactory>
```

7.2. Server-side interceptor

JORAM allows some special objects called Interceptors to handle messages during their transit in the MOM, there are two types of interceptors:

- An user's interceptors receives each message that's entering or exiting the server.
- A destination's interceptor receives each message sent to this destination.

Interceptors can read and modify the messages, this enables filtering, transformation or content enrichment, for example adding a property into the message. Interceptors can also stop the Interceptor chain by simply returning false to their intercept method invocation, in this case the transmission of the message is stopped.

7.2.1. Writing an interceptor

Writing a server-side interceptor is quite easy, just create a class that implements the `MessageInterceptor`³ interface (see below). The only method to implement is `handleMessage`, this method returns a boolean result. If it is true the server continue to process the interceptor chain, if it is false the server stops executing the chain and removes the message (the message is sent to the corresponding DMQ if any).

The method parameter is a `Message`⁴ object containing both the header and the payload of the corresponding message. Each interceptor can observe the state of the message and modify it. From the JORAM server point of view the interceptors are stateless, If you need to keep data you must do it yourself.

All interceptors class must be accessible by the class loader of the JORAM server (possibly dynamically load in the case of a server running on OSGi).

7.2.2. MessageInterceptor Interface

Interface `org.objectweb.joram.mom.util.MessageInterceptor`

Public Member Functions

`void handle (Message message)`

Detailed Description

Session level message interceptor interface.

The `MessageInterceptor` should be implemented by any class whose instances are intended to intercept either or both following operations within a JMS Session:

- sending a message
- receiving a message

The interceptor can be attached to a JMS Session through configuration (see `joramAdmin.xml` or `FactoryParameters` class) as IN (when consuming a message) or OUT (when producing a message) interceptor.

Member Function Documentation

void `MessageInterceptor.handle (Message message)`

This method handles a message before proceeding. convention, the implementation can modify the original message or the current runtime context, and return no out value. It also avoids to throw any exception within this method.

Parameters:

<code>message</code>	the message to handle.
----------------------	------------------------

7.2.3. Configuring interceptors on destination

The interceptor chain is activated on each message that reaches the destination. This processing is done before storing the message for a queue destination or before forwarding the message for a topic one.

Initial configuration

The interceptor chains can be simply configured at the creation of a the destination using either the administration API or the XML scripting language. It only needs to defines the corresponding property (`jms_joram_interceptors`) with the comma separated list of the classname of your interceptors.

³ Package `org.objectweb.joram.mom.util`

⁴ Package `org.objectweb.joram.shared.messages`

Using administration API, you simply have to call the appropriate create method of the Queue or Topic administration class with a properties object defining the `jms_joram_interceptors` property. For example for a Queue:

```
Properties properties = new Properties();
properties.setProperty("jms_joram_interceptors",
                      "interceptors classname list");
Queue.create(..., properties, ...);
```

In order to configure the interceptors of a destination in a XML script you simply have to add the definition of the `"jms_joram_interceptors"` property in the destination declaration. For example for a Queue:

```
<Queue name="queue">
  <property name="jms_joram_interceptors" value="interceptors class name list"/>
...
</Queue>
```

Administration changes

Once the destination is created you can always change the interceptor's chains, adding a list of interceptors, removing or replacing some of them. The Queue and Topic class of the administration API offer the following methods:

- `public String getInterceptors();`
- `public void addInterceptors(String interceptors);`
- `public void removeInterceptors(String interceptors);`
- `public void replaceInterceptor(String newInterceptor, String oldInterceptor);`

An AdminException is thrown if the destination does not exist on the server side. A ConnectException is thrown if the admin connection with the server is closed or lost.

7.2.4. Configuring interceptors on user's connections

There are two distinct chains of interceptors. The first one `"interceptors_in"` handles each message that's entering the server (result of a send method on a connection from the selected user). The second one `"interceptors_out"` handles each message that's exiting the server (result of a receive method on a connection from the selected user). These two interceptor chains are configurable for each user.

Initial configuration

The interceptor chains can be simply configured at the creation of a user using either the administration API or the XML scripting language. It only needs to define the corresponding property (`jms_joram_interceptors_in` or `jms_joram_interceptors_out`) with the comma separated list of the classname of your interceptors.

Using administration API, you simply have to call the appropriate create method of the User administration class with a properties object defining the `jms_joram_interceptors_in` and `jms_joram_interceptors_out` properties:

```
Properties properties = new Properties();
properties.setProperty("jms_joram_interceptors_in",
                      "interceptors classname list");
properties.setProperty("jms_joram_interceptors_out",
                      "interceptors classname list");
User.create(..., properties);
```

In order to configure the interceptors of an user in a XML script you simply have to add the definition of the "jms_joram_interceptors_in" and/or "jms_joram_interceptors_out" properties in the user declaration:

```
<User name="anonymous" password="anonymous">
    <property name="jms_joram_interceptors_in"
              value="interceptors classname list"/>
    <property name="jms_joram_interceptors_out"
              value="interceptors classname list"/>
    ...
</User>
```

Administration changes

Once the user is created you can always change the interceptor's chains, adding a list of interceptors, removing or replacing some of them. The User class of the administration API offers the following methods:

- public String getInterceptorsIN();
- public void addInterceptorsIN(String interceptors);
- public void removeInterceptorsIN(String interceptors);
- public void replaceInterceptorIN(String newInterceptor, String oldInterceptor);
-
- public String getInterceptorsOUT();
- public void addInterceptorsOUT(String interceptors);
- public void removeInterceptorsOUT(String interceptors);
- public void replaceInterceptorOUT(String newInterceptor, String oldInterceptor);

An AdminException is thrown if the user agent does not exist on the server side. A ConnectException is thrown if the administration connection with the server is closed or lost.

8. PTP JMS API

Class Reference

This chapter remind notions of the old JMS 1.0 API dedicated to Point-To-Point interface.

8.1. QueueLocalConnectionFactory

Class org.objectweb.joram.client.jms.local.QueueLocalConnectionFactory
 Extends org.objectweb.joram.client.jms.QueueConnectionFactory

Static Public Member Functions

static javax.jms.QueueConnectionFactory create ()

Detailed Description

A QueueLocalConnectionFactory instance is a factory of local connections for PTP communication.

Deprecated:

Replaced next to Joram 5.2.1 by LocalConnectionFactory.

Member Function Documentation

static javax.jms.QueueConnectionFactory QueueLocalConnectionFactory.create () [static]

Administration method creating a javax.jms.QueueConnectionFactory instance for creating local connections.

8.2. QueueTcpConnectionFactory

Class org.objectweb.joram.client.jms.tcp.QueueTcpConnectionFactory
 Extends org.objectweb.joram.client.jms.QueueConnectionFactory

Static Public Member Functions

static javax.jms.QueueConnectionFactory create ()

static javax.jms.QueueConnectionFactory create (String host, int port)

static javax.jms.QueueConnectionFactory create (String host, int port, String reliableClass)

Detailed Description

A QueueTcpConnectionFactory instance is a factory of TCP connections for PTP communication.

Deprecated:

Replaced next to Joram 5.2.1 by TcpConnectionFactory.

Member Function Documentation

static javax.jms.QueueConnectionFactory QueueTcpConnectionFactory.create () [static]

Administration method creating a javax.jms.QueueConnectionFactory instance for creating TCP connections with the default server.

Exceptions:

ConnectException	If the admin connection is closed or broken.
------------------	--

See also:

getDefaultServerHost()
getDefaultServerPort()

static javax.jms.QueueConnectionFactory QueueTcpConnectionFactory.create (String host, int port) [static]

Administration method creating a javax.jms.QueueConnectionFactory instance for creating TCP connections with a given server.

Parameters:

host	Name or IP address of the server's host.
port	Server's listening port.

static javax.jms.QueueConnectionFactory QueueTcpConnectionFactory.create (String host, int port, String reliableClass) [static]

Administration method creating a javax.jms.QueueConnectionFactory instance for creating TCP connections with a given server.

Parameters:

host	Name or IP address of the server's host.
port	Server's listening port.
reliableClass	Reliable class name.

8.3. QueueConnection

Class org.objectweb.joram.client.jms.QueueConnection
 Extends org.objectweb.joram.client.jms.Connection
 Implements javax.jms.QueueConnection

Public Member Functions

javax.jms.ConnectionConsumer createConnectionConsumer (javax.jms.Queue queue, String selector, javax.jms.ServerSessionPool sessionPool, int maxMessages) throws JMSEException
 synchronized javax.jms.QueueSession createQueueSession (boolean transacted, int acknowledgeMode) throws JMSEException

Detailed Description

Implements the javax.jms.QueueConnection interface.

Member Function Documentation

javax.jms.ConnectionConsumer QueueConnection.createConnectionConsumer (javax.jms.Queue queue, String selector, javax.jms.ServerSessionPool sessionPool, int maxMessages) throws JMSEException

API method. Creates a connection consumer for this connection, this is an expert facility needed for applications servers.

Parameters:

queue	the queue to access.
selector	only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for this message consumer.
sessionPool	the server session pool to associate with this connection consumer
maxMessages	the maximum number of messages that can be assigned to a server session at one time.

Returns:

The connection consumer.

Exceptions:

IllegalStateException	If the connection is closed.
InvalidSelectorException	If the selector syntax is wrong.
InvalidDestinationException	If the target destination does not exist.
JMSEException	If the method fails for any other reason.

synchronized javax.jms.QueueSession QueueConnection.createQueueSession (boolean transacted, int acknowledgeMode) throws JMSEException

API method. Creates a QueueSession object.

Parameters:

transacted	indicates whether the session is transacted.
acknowledgeMode	indicates whether the consumer or the client will acknowledge any messages it receives; ignored if the session is transacted. Legal values are Session.AUTO_ACKNOWLEDGE, Session.CLIENT_ACKNOWLEDGE, and Session.DUPS_OK_ACKNOWLEDGE.

Returns:

A newly created session.

Exceptions:

IllegalStateException	If the connection is closed.
JMSEException	In case of an invalid acknowledge mode.

8.4. QueueSession

Class org.objectweb.joram.client.jms.QueueSession
 Extends org.objectweb.joram.client.jms.Session
 Implements javax.jms.QueueSession

Public Member Functions

synchronized javax.jms.QueueSender createSender (javax.jms.Queue queue) throws JMSEException
 javax.jms.QueueReceiver createReceiver (javax.jms.Queue queue, String selector) throws JMSEException
 javax.jms.QueueReceiver createReceiver (javax.jms.Queue queue) throws JMSEException

Detailed Description

Implements the javax.jms.QueueSession interface.

Member Function Documentation

javax.jms.QueueReceiver QueueSession.createReceiver (javax.jms.Queue queue, String selector) throws JMSEException

API method. Creates a QueueReceiver object to receive messages from the specified queue using a message selector.

Parameters:

queue	the queue to access.
selector	The selector allowing to filter messages.

Returns:

the created QueueReceiver object.

Exceptions:

IllegalStateException	If the session is closed or if the connection is broken.
JMSEException	If the creation fails for any other reason.

javax.jms.QueueReceiver QueueSession.createReceiver (javax.jms.Queue queue) throws JMSEException

API method. Creates a QueueReceiver object to receive messages from the specified queue.

Parameters:

queue	the Queue to access.
-------	----------------------

Returns:

the created QueueReceiver object.

Exceptions:

IllegalStateException	If the session is closed or if the connection is broken.
JMSEException	If the creation fails for any other reason.

synchronized javax.jms.QueueSender QueueSession.createSender (javax.jms.Queue queue) throws JMSEException

API method. Creates a QueueSender object to send messages to the specified queue.

Parameters:

queue	the Queue to send to, or null if this is a sender which does not have a specified destination.
-------	--

Returns:

the created QueueSender object.

Exceptions:

IllegalStateException	If the session is closed or if the connection is broken.
JMSEException	If the creation fails for any other reason.

8.5. QueueReceiver

Class org.objectweb.joram.client.jms.QueueReceiver
 Extends org.objectweb.joram.client.jms.MessageConsumer
 Implements javax.jms.QueueReceiver

Public Member Functions

`javax.jms.Queue getQueue () throws JMSException`

Detailed Description

Implements the `javax.jms.QueueReceiver` interface.

Member Function Documentation

`javax.jms.Queue org.objectweb.joram.client.jms.QueueReceiver.getQueue () throws JMSException`

API method. Gets the queue associated with this queue receiver.

Returns:

this receiver's Queue

Exceptions:

<code>IllegalStateException</code>	If the receiver is closed.
------------------------------------	----------------------------

8.6. QueueSender

Class `org.objectweb.joram.client.jms.QueueSender`

Extends `org.objectweb.joram.client.jms.MessageProducer`

Implements `javax.jms.QueueSender`

Public Member Functions

`javax.jms.Queue getQueue () throws JMSException`

`void send (javax.jms.Queue queue, javax.jms.Message message) throws JMSException`

`void send (javax.jms.Queue queue, javax.jms.Message message, int deliveryMode, int priority, long timeToLive)`
throws `JMSException`

Detailed Description

Implements the `javax.jms.QueueSender` interface.

Member Function Documentation

`javax.jms.Queue QueueSender.getQueue () throws JMSException`

API method. Gets the queue associated with this queue sender.

Returns:

this sender's Queue.

Exceptions:

<code>IllegalStateException</code>	If the sender is closed.
------------------------------------	--------------------------

`void QueueSender.send (javax.jms.Queue queue, javax.jms.Message message) throws JMSException`

API method. Sends a message to a queue for an unidentified queue sender with default delivery parameters.

Typically, a queue sender is assigned a queue at creation time; however, the JMS API also supports unidentified queue sender, which require that the queue be supplied every time a message is sent.

Parameters:

queue	the queue to send this message to.
message	the message to send.

Exceptions:

UnsupportedOperationException	When the sender did not properly identify itself.
JMSecurityException	If the user is not a WRITER on the specified queue.
IllegalStateException	If the sender is closed, or if the connection is broken.
JMSEException	If the request fails for any other reason.

void QueueSender.send (javax.jms.Queue queue, javax.jms.Message message, int deliveryMode, int priority, long timeToLive) throws JMSEException

API method. Sends a message to a queue for an unidentified queue sender with given delivery parameters. Typically, a queue sender is assigned a queue at creation time; however, the JMS API also supports unidentified queue sender, which require that the queue be supplied every time a message is sent.

Parameters:

queue	the queue to send this message to.
message	the message to send.
deliveryMode	the delivery mode to use.
priority	the priority for this message.
timeToLive	the message's lifetime in milliseconds.

Exceptions:

UnsupportedOperationException	When the sender did not properly identify itself.
JMSecurityException	If the user is not a WRITER on the specified queue.
IllegalStateException	If the sender is closed, or if the connection is broken.
JMSEException	If the request fails for any other reason.

8.7. XAQueueLocalConnectionFactory

Class org.objectweb.joram.client.jms.local.XAQueueLocalConnectionFactory
Extends org.objectweb.joram.client.jms.XAQueueConnectionFactory

Static Public Member Functions

static javax.jms.XAQueueConnectionFactory create ()

Detailed Description

An XAQueueLocalConnectionFactory instance is a factory of local connections for XA PTP communication.

Deprecated:

Replaced next to Joram 5.2.1 by LocalConnectionFactory.

Member Function Documentation

static javax.jms.XAQueueConnectionFactory XAQueueLocalConnectionFactory.create () [static]

Administration method creating a javax.jms.XAQueueConnectionFactory instance for creating local connections.

8.8. XAQueueTcpConnectionFactory

Class org.objectweb.joram.client.jms.tcp.XAQueueTcpConnectionFactory

Extends org.objectweb.joram.client.jms.XAQueueConnectionFactory

Static Public Member Functions

```
static javax.jms.XAQueueConnectionFactory create () throws java.net.ConnectException
static javax.jms.XAQueueConnectionFactory create (String host, int port)
static javax.jms.XAQueueConnectionFactory create (String host, int port, String reliableClass)
```

Detailed Description

An XAQueueTcpConnectionFactory instance is a factory of TCP connections for XA PTP communication.

Deprecated:

Replaced next to Joram 5.2.1 by TcpConnectionFactory.

Member Function Documentation

static javax.jms.XAQueueConnectionFactory XAQueueTcpConnectionFactory.create () throws java.net.ConnectException [static]

Administration method creating a javax.jms.XAQueueConnectionFactory instance for creating TCP connections with the default server.

Exceptions:

java.net.ConnectException	If the admin connection is closed or broken.
---------------------------	--

See also:

getDefaultServerHost()
getDefaultServerPort()

static javax.jms.XAQueueConnectionFactory XAQueueTcpConnectionFactory.create (String host, int port) [static]

Administration method creating a javax.jms.XAQueueConnectionFactory instance for creating TCP connections with a given server.

Parameters:

host	Name or IP address of the server's host.
port	Server's listening port.

static javax.jms.XAQueueConnectionFactory XAQueueTcpConnectionFactory.create (String host, int port, String reliableClass) [static]

Administration method creating a javax.jms.XAQueueConnectionFactory instance for creating TCP connections with a given server.

Parameters:

host	Name or IP address of the server's host.
port	Server's listening port.
reliableClass	Reliable class name.

8.9. XAQueueConnection

Class org.objectweb.joram.client.jms.XAQueueConnection
 Extends org.objectweb.joram.client.jms.QueueConnection
 Implements javax.jms.XAQueueConnection

Public Member Functions

`javax.jms.QueueSession createQueueSession (boolean transacted, int acknowledgeMode)` throws `JMSEException`
`javax.jms.XAQueueSession createXAQueueSession ()` throws `JMSEException`

Detailed Description

Implements the `javax.jms.XAQueueConnection` interface.

Member Function Documentation

`javax.jms.QueueSession XAQueueConnection.createQueueSession (boolean transacted, int acknowledgeMode) throws JMSEException`

API method. Creates a `QueueSession` object.

Parameters:

<code>transacted</code>	indicates whether the session is transacted.
<code>acknowledgeMode</code>	indicates whether the consumer or the client will acknowledge any messages it receives; ignored if the session is transacted. Legal values are <code>Session.AUTO_ACKNOWLEDGE</code> , <code>Session.CLIENT_ACKNOWLEDGE</code> , and <code>Session.DUPS_OK_ACKNOWLEDGE</code> .

Returns:

A newly created session.

Exceptions:

<code>IllegalStateException</code>	If the connection is closed.
<code>JMSEException</code>	In case of an invalid acknowledge mode.

`javax.jms.XAQueueSession XAQueueConnection.createXAQueueSession () throws JMSEException`

API method. Creates an `XAQueueSession` object.

Returns:

A newly created session.

Exceptions:

<code>IllegalStateException</code>	If the connection is closed.
------------------------------------	------------------------------

8.10. XAQueueSession

Class `org.objectweb.joram.client.jms.XAQueueSession`
Extends `org.objectweb.joram.client.jms.XASession`
Implements `javax.jms.XAQueueSession`

Public Member Functions

`javax.jms.QueueSession getQueueSession ()` throws `JMSEException`

Detailed Description

Implements the `javax.jms.XAQueueSession` interface.

Member Function Documentation

javax.jms.QueueSession XAQueueSession.getQueueSession () throws JMSException

API method. Gets the queue session associated with this XAQueueSession.

Returns:

the queue session object.

Exceptions:

JMSException	if an internal error occurs.
--------------	------------------------------

9. P/S JMS API

Class Reference

This chapter remind notions of the old JMS 1.0 API dedicated to Publish/Subscribe interface.

9.1. *TopicLocalConnectionFactory*

Class org.objectweb.joram.client.jms.local.TopicLocalConnectionFactory
 Extends org.objectweb.joram.client.jms.TopicConnectionFactory

Static Public Member Functions

static javax.jms.TopicConnectionFactory create ()

Detailed Description

A TopicLocalConnectionFactory instance is a factory of local connections for Pub/Sub communication.

Deprecated:

Replaced next to Joram 5.2.1 by LocalConnectionFactory.

Member Function Documentation

static javax.jms.TopicConnectionFactory TopicLocalConnectionFactory.create () [static]

Administration method creating a javax.jms.TopicConnectionFactory instance for creating local connections.

9.2. *TopicTcpConnectionFactory*

Class org.objectweb.joram.client.jms.tcp.TopicTcpConnectionFactory
 Extends org.objectweb.joram.client.jms.TopicConnectionFactory

Static Public Member Functions

static javax.jms.TopicConnectionFactory create () throws java.net.ConnectException
 static javax.jms.TopicConnectionFactory create (String host, int port)
 static javax.jms.TopicConnectionFactory create (String host, int port, String reliableClass)

Detailed Description

A TopicTcpConnectionFactory instance is a factory of TCP connections for Pub/Sub communication.

Deprecated:

Replaced next to Joram 5.2.1 by TcpConnectionFactory.

Member Function Documentation

static javax.jms.TopicConnectionFactory TopicTcpConnectionFactory.create () throws java.net.ConnectException [static]

Administration method creating a javax.jms.TopicConnectionFactory instance for creating TCP connections with the default server.

Exceptions:

java.net.ConnectException	If the admin connection is closed or broken.
---------------------------	--

See also:

getDefaultServerHost()
getDefaultServerPort()

static javax.jms.TopicConnectionFactory TopicTcpConnectionFactory.create (String host, int port) [static]

Administration method creating a javax.jms.TopicConnectionFactory instance for creating TCP connections with a given server.

Parameters:

host	Name or IP address of the server's host.
port	Server's listening port.

static javax.jms.TopicConnectionFactory TopicTcpConnectionFactory.create (String host, int port, String reliableClass) [static]

Administration method creating a javax.jms.TopicConnectionFactory instance for creating TCP connections with a given server.

Parameters:

host	Name or IP address of the server's host.
port	Server's listening port.
reliableClass	Reliable class name.

9.3. *TopicConnection*

Class org.objectweb.joram.client.jms.TopicConnection

Public Member Functions

javax.jms.ConnectionConsumer createConnectionConsumer (javax.jms.Topic topic, String selector, javax.jms.ServerSessionPool sessionPool, int maxMessages) throws JMSEException
javax.jms.TopicSession createTopicSession (boolean transacted, int acknowledgeMode) throws JMSEException

Detailed Description

Implements the javax.jms.TopicConnection interface.

Member Function Documentation

javax.jms.ConnectionConsumer TopicConnection.createConnectionConsumer (javax.jms.Topic topic, String selector, javax.jms.ServerSessionPool sessionPool, int maxMessages) throws JMSEException

API method. Creates a connection consumer for this connection, this is an expert facility needed for applications servers.

Parameters:

topic	the topic to access.
selector	only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for this message consumer.
sessionPool	the server session pool to associate with this connection consumer
maxMessages	the maximum number of messages that can be assigned to a server session at one time.

Returns:

The connection consumer.

Exceptions:

IllegalStateException	If the connection is closed.
InvalidSelectorException	If the selector syntax is wrong.
InvalidDestinationException	If the target destination does not exist.
JMSEException	If the method fails for any other reason.

javax.jms.TopicSession TopicConnection.createTopicSession (boolean transacted, int acknowledgeMode) throws JMSEException

API method. Creates a TopicSession object.

Parameters:

transacted	indicates whether the session is transacted.
acknowledgeMode	indicates whether the consumer or the client will acknowledge any messages it receives; ignored if the session is transacted. Legal values are Session.AUTO_ACKNOWLEDGE, Session.CLIENT_ACKNOWLEDGE, and Session.DUPS_OK_ACKNOWLEDGE.

Returns:

A newly created session.

Exceptions:

IllegalStateException	If the connection is closed.
JMSEException	In case of an invalid acknowledge mode.

9.4. TopicSession

Class org.objectweb.joram.client.jms.TopicSession
 Extends org.objectweb.joram.client.jms.Session
 Implements javax.jms.TopicSession

Public Member Functions

javax.jms.TopicPublisher createPublisher (javax.jms.Topic topic) throws JMSEException
 javax.jms.TopicSubscriber createSubscriber (javax.jms.Topic topic, String selector, boolean noLocal) throws JMSEException
 javax.jms.TopicSubscriber createSubscriber (javax.jms.Topic topic) throws JMSEException

Detailed Description

Implements the javax.jms.TopicSession interface.

Member Function Documentation

javax.jms.TopicPublisher TopicSession.createPublisher (javax.jms.Topic topic) throws JMSEException

API method. Creates a TopicPublisher object to send messages to the specified topic.

Parameters:

topic	the topic to access.
-------	----------------------

Returns:

the created TopicPublisher object.

Exceptions:

IllegalStateException	If the session is closed or if the connection is broken.
JMSEException	If the creation fails for any other reason.

javax.jms.TopicSubscriber TopicSession.createSubscriber (javax.jms.Topic topic, String selector, boolean noLocal) throws JMSEException

API method. Creates a TopicSubscriber object to receive messages from the specified topic using a message selector. The subscriber NoLocal attribute allows a subscriber to inhibit the delivery of messages published by its own connection. The default value for this attribute is false.

Parameters:

topic	the topic to access.
selector	The selector allowing to filter messages.
noLocal	if set, inhibits the delivery of messages published by its own connection.

Returns:

the created TopicSubscriber object.

Exceptions:

IllegalStateException	If the session is closed or if the connection is broken.
JMSEException	If the creation fails for any other reason.

javax.jms.TopicSubscriber TopicSession.createSubscriber (javax.jms.Topic topic) throws JMSEException

API method. Creates a TopicSubscriber object to receive messages from the specified topic.

Parameters:

topic	the topic to access.
-------	----------------------

Returns:

the created TopicSubscriber object.

Exceptions:

IllegalStateException	If the session is closed or if the connection is broken.
JMSEException	If the creation fails for any other reason.

9.5. TopicPublisher

Class org.objectweb.joram.client.jms.TopicPublisher
 Extends org.objectweb.joram.client.jms.MessageProducer
 Implements javax.jms.TopicPublisher

Public Member Functions

```
javax.jms.Topic getTopic () throws JMSException
void publish (javax.jms.Message message, int deliveryMode, int priority, long timeToLive) throws JMSException
void publish (javax.jms.Message message) throws JMSException
void publish (javax.jms.Topic topic, javax.jms.Message message) throws JMSException
void publish (javax.jms.Topic topic, javax.jms.Message message, int deliveryMode, int priority, long timeToLive)
throws JMSException
```

Detailed Description

Implements the javax.jms.TopicPublisher interface.

Member Function Documentation

javax.jms.Topic TopicPublisher.getTopic () throws JMSException

API method. Gets the topic associated with this topic publisher.

Returns:

this publisher's topic.

Exceptions:

IllegalStateException	If the publisher is closed.
-----------------------	-----------------------------

void TopicPublisher.publish (javax.jms.Message message, int deliveryMode, int priority, long timeToLive) throws JMSException

API method. Sends a message to a topic with given delivery parameters.

Parameters:

message	the message to send.
deliveryMode	the delivery mode to use.
priority	the priority for this message.
timeToLive	the message's lifetime in milliseconds.

Exceptions:

IllegalStateException	If the publisher is closed, or if the connection is broken.
JMSException	If the request fails for any other reason.

void TopicPublisher.publish (javax.jms.Message message) throws JMSException

API method. Sends a message to a topic with default delivery parameters.

Parameters:

message	the message to send.
---------	----------------------

Exceptions:

IllegalStateException	If the publisher is closed, or if the connection is broken.
JMSException	If the request fails for any other reason.

void TopicPublisher.publish (javax.jms.Topic topic, javax.jms.Message message) throws JMSException

API method. Sends a message to a topic for an unidentified topic publisher with default delivery parameters.

Typically, a topic publisher is assigned a topic at creation time; however, the JMS API also supports unidentified topic publisher, which require that the topic be supplied every time a message is sent.

Parameters:

topic	the topic to send this message to.
-------	------------------------------------

message	the message to send.
---------	----------------------

Exceptions:

IllegalStateException	If the publisher is closed, or if the connection is broken.
JMSEException	If the request fails for any other reason.

void TopicPublisher.publish (javax.jms.Topic topic, javax.jms.Message message, int deliveryMode, int priority, long timeToLive) throws JMSEException

API method. Sends a message to a topic for an unidentified topic publisher with given delivery parameters. Typically, a topic publisher is assigned a topic at creation time; however, the JMS API also supports unidentified topic publisher, which require that the topic be supplied every time a message is sent.

Parameters:

topic	the topic to send this message to.
message	the message to send.
deliveryMode	the delivery mode to use.
priority	the priority for this message.
timeToLive	the message's lifetime in milliseconds.

Exceptions:

IllegalStateException	If the publisher is closed, or if the connection is broken.
JMSEException	If the request fails for any other reason.

9.6. TopicSubscriber

Class org.objectweb.joram.client.jms.TopicSubscriber
 Extends org.objectweb.joram.client.jms.MessageConsumer
 Implements javax.jms.TopicSubscriber

Public Member Functions

boolean getNoLocal () throws JMSEException
 javax.jms.Topic getTopic () throws JMSEException

Detailed Description

Implements the javax.jms.TopicSubscriber interface.

Member Function Documentation

boolean TopicSubscriber.getNoLocal () throws JMSEException

API method. Gets the NoLocal attribute for this subscriber. The default value for this attribute is false.

Returns:

true if locally published messages are being inhibited

Exceptions:

IllegalStateException	If the subscriber is closed.
-----------------------	------------------------------

javax.jms.Topic TopicSubscriber.getTopic () throws JMSEException

API method. Gets the topic associated with this topic publisher.

Returns:

this subscriber's topic.

Exceptions:

IllegalStateException	If the subscriber is closed.
-----------------------	------------------------------

9.7. XATopicLocalConnectionFactory

Class org.objectweb.joram.client.jms.local.XATopicLocalConnectionFactory
 Extends org.objectweb.joram.client.jms.XATopicConnectionFactory

Static Public Member Functions

static javax.jms.XATopicConnectionFactory create ()

Detailed Description

An XATopicLocalConnectionFactory instance is a factory of local connections for XA Pub/Sub communication.

Deprecated:

Replaced next to Joram 5.2.1 by LocalConnectionFactory.

Member Function Documentation

static javax.jms.XATopicConnectionFactory XATopicLocalConnectionFactory.create () [static]

Administration method creating a javax.jms.XATopicConnectionFactory instance for creating local connections.

9.8. XATopicTcpConnectionFactory

Class org.objectweb.joram.client.jms.tcp.XATopicTcpConnectionFactory
 Extends org.objectweb.joram.client.jms.XATopicConnectionFactory

Static Public Member Functions

static javax.jms.XATopicConnectionFactory create () throws java.net.ConnectException
 static javax.jms.XATopicConnectionFactory create (String host, int port)
 static javax.jms.XATopicConnectionFactory create (String host, int port, String reliableClass)

Detailed Description

An XATopicTcpConnectionFactory instance is a factory of TCP connections for XA Pub/Sub communication.

Deprecated:

Replaced next to Joram 5.2.1 by TcpConnectionFactory.

Member Function Documentation

static javax.jms.XATopicConnectionFactory XATopicTcpConnectionFactory.create () throws java.net.ConnectException [static]

Administration method creating a javax.jms.XATopicConnectionFactory instance for creating TCP connections with the default server.

Exceptions:

java.net.ConnectException	If the admin connection is closed or broken.
---------------------------	--

See also:

`getDefaultServerHost()`
`getDefaultServerPort()`

static javax.jms.XATopicConnectionFactory XATopicTcpConnectionFactory.create (String host, int port) [static]

Administration method creating a `javax.jms.XATopicConnectionFactory` instance for creating TCP connections with a given server.

Parameters:

host	Name or IP address of the server's host.
port	Server's listening port.

static javax.jms.XATopicConnectionFactory XATopicTcpConnectionFactory.create (String host, int port, String reliableClass) [static]

Administration method creating a `javax.jms.XATopicConnectionFactory` instance for creating TCP connections with a given server.

Parameters:

host	Name or IP address of the server's host.
port	Server's listening port.
reliableClass	Reliable class name.

9.9. XATopicConnection

Class `org.objectweb.joram.client.jms.XATopicConnection`

Extends `org.objectweb.joram.client.jms.TopicConnection`

Implements `javax.jms.XATopicConnection`

Public Member Functions

`javax.jms.TopicSession createTopicSession (boolean transacted, int acknowledgeMode) throws JMSEException`
`javax.jms.XATopicSession createXATopicSession () throws JMSEException`

Detailed Description

Implements the `javax.jms.XATopicConnection` interface.

Member Function Documentation

javax.jms.TopicSession XATopicConnection.createTopicSession (boolean transacted, int acknowledgeMode) throws JMSEException

API method. Creates a `TopicSession` object.

Parameters:

transacted	indicates whether the session is transacted.
acknowledgeMode	indicates whether the consumer or the client will acknowledge any messages it receives; ignored if the session is transacted. Legal values are <code>Session.AUTO_ACKNOWLEDGE</code> , <code>Session.CLIENT_ACKNOWLEDGE</code> , <code>Session.DUPS_OK_ACKNOWLEDGE</code> . and

Returns:

A newly created session.

Exceptions:

IllegalStateException	If the connection is closed.
JMSEException	In case of an invalid acknowledge mode.

javax.jms.XATopicSession XATopicConnection.createXATopicSession () throws JMSEException

API method. Creates an XATopicSession object.

Returns:

A newly created session.

Exceptions:

IllegalStateException	If the connection is closed.
-----------------------	------------------------------

9.10. XATopicSession

Class org.objectweb.joram.client.jms.XATopicSession

Extends org.objectweb.joram.client.jms.XASession

Implements javax.jms.XATopicSession

Public Member Functions

javax.jms.TopicSession getTopicSession () throws JMSEException

Detailed Description

Implements the javax.jms.XATopicSession interface.

Member Function Documentation

javax.jms.TopicSession XATopicSession.getTopicSession () throws JMSEException

API method. Gets the topic session associated with this XATopicSession.

Returns:

the topic session object.

Exceptions:

JMSEException	if an internal error occurs.
---------------	------------------------------

10. Application Server Class Reference

This chapter is devoted to the API needed for application servers, it describes expert facilities needed for applications servers.

10.1. MultiSessionConsumer

Class org.objectweb.joram.client.jms.MultiSessionConsumer
Implements org.objectweb.joram.client.jms.MessageConsumerListener

Detailed Description

The MultiSessionConsumer is threaded (see MessageDispatcher) because the session pool can hang if there is no more available ServerSession.

11. Administration API

Class Reference

11.1. AdminException

Class org.objectweb.joram.client.jms.admin.AdminException

Inherited by

- org.objectweb.joram.client.jms.admin.NameAlreadyUsedException,
- org.objectweb.joram.client.jms.admin.ServerIdAlreadyUsedException,
- org.objectweb.joram.client.jms.admin.StartFailureException, and
- org.objectweb.joram.client.jms.admin.UnknownServerException.

Detailed Description

An AdminException is an exception thrown by an administration method.

Constructor Documentation

`AdminException (String info)`

Constructs an AdminException instance.

Parameters:

info	Information about the exception.
------	----------------------------------

11.2. AdminHelper

Class org.objectweb.joram.client.jms.admin.AdminHelper

Static Public Member Functions

```
static void setClusterLink (Topic clusterTopic, Topic joiningTopic) throws ConnectException, AdminException
static void unsetClusterLink (Topic topic) throws ConnectException, AdminException
static void setHierarchicalLink (Topic father, Topic son) throws ConnectException, AdminException
static void unsetHierarchicalLink (Topic topic) throws ConnectException, AdminException
static void setQueueCluster (Queue clusterQueue, Queue joiningQueue) throws ConnectException,
AdminException
static void setQueueCluster (Destination clusterQueue, Queue joiningQueue) throws ConnectException,
AdminException
static void leaveQueueCluster (Queue clusterQueue, Queue leaveQueue) throws ConnectException,
AdminException
static AdminReply listQueueCluster (Queue clusterQueue) throws ConnectException, AdminException
```

Detailed Description

The AdminHelper class is a utility class providing methods for building special configurations such as topics cluster or hierarchy, queues cluster, etc.

Deprecated:

Member Function Documentation

static void AdminHelper.leaveQueueCluster (Queue clusterQueue, Queue leaveQueue) throws ConnectException, AdminException [static]

Removes a queue from the cluster Queue it is part of.

The request fails if the queue does not exist or is not part of any cluster.

Parameters:

clusterQueue	the cluster Queue.
leaveQueue	Queue leaving the cluster Queue it is part of.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

Deprecated:

static AdminReply AdminHelper.listQueueCluster (Queue clusterQueue) throws ConnectException, AdminException [static]

List a cluster queue.

Parameters:

clusterQueue	the cluster Queue.
--------------	--------------------

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

Deprecated:

static void AdminHelper.setClusterLink (Topic clusterTopic, Topic joiningTopic) throws ConnectException, AdminException [static]

Links two given topics in a cluster relationship.

The request fails if one or both of the topics are deleted, or can't belong to a cluster.

Parameters:

clusterTopic	Topic part of the cluster, or chosen as the initiator of the cluster.
joiningTopic	Topic joining the cluster.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

Deprecated:

static void setHierarchicalLink (Topic father, Topic son) throws ConnectException, AdminException [static]

Links two given topics in a hierarchical relationship.

The request fails if one of the topics does not exist or can't be part of a hierarchy.

Parameters:

father	Father.
son	Son.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

Deprecated:

static void AdminHelper.setQueueCluster (Queue clusterQueue, Queue joiningQueue) throws ConnectException, AdminException [static]

Adds a queue to a cluster.

The request fails if one or both of the queues are deleted, or can't belong to a cluster.

Parameters:

clusterQueue	Queue part of the cluster, or chosen as the initiator of the cluster.
joiningQueue	Queue joining the cluster.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

Deprecated:

static void AdminHelper.setQueueCluster (Destination clusterQueue, Queue joiningQueue) throws ConnectException, AdminException [static]

Adds a queue to a cluster.

The request fails if one or both of the queues are deleted, or can't belong to a cluster.

Parameters:

clusterQueue	Queue part of the cluster, or chosen as the initiator of the cluster.
joiningQueue	Queue joining the cluster.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

Deprecated:

static void AdminHelper.unsetClusterLink (Topic topic) throws ConnectException, AdminException [static]

Removes a topic from the cluster it is part of.

The request fails if the topic does not exist or is not part of any cluster.

Parameters:

topic	Topic leaving the cluster it is part of.
-------	--

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

Deprecated:

static void AdminHelper.unsetHierarchicalLink (Topic topic) throws ConnectException, AdminException [static]

Unsets the father of a given topic.

The request fails if the topic does not exist or is not part of any hierarchy.

Parameters:

topic	Topic which father is unset.
-------	------------------------------

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

Deprecated:

11.3. AdminItf

Interface org.objectweb.joram.client.jms.admin.AdminItf

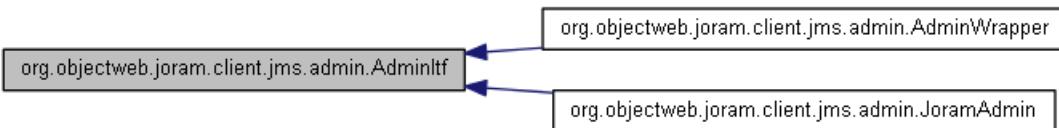


Figure 11.1 - Inheritance diagram for AdminItf interface

Public Member Functions

```

void setTimeOutToAbortRequest (long timeOut) throws ConnectException
long getTimeOutToAbortRequest () throws ConnectException
void close ()
boolean isClosed ()
void stopServer () throws ConnectException, AdminException
void stopServer (int serverId) throws ConnectException, AdminException
void addServer (int sid, String host, String domain, int port, String server) throws ConnectException, AdminException
void addServer (int sid, String host, String domain, int port, String server, String[] services, String[] args) throws ConnectException, AdminException
void removeServer (int sid) throws ConnectException, AdminException
void addDomain (String domain, int sid, int port) throws ConnectException, AdminException
void addDomain (String domain, String network, int sid, int port) throws ConnectException, AdminException
void removeDomain (String domain) throws ConnectException, AdminException
String getConfiguration () throws ConnectException, AdminException
Hashtable getStatistics () throws ConnectException, AdminException
Hashtable getStatistics (int serverId) throws ConnectException, AdminException
String getDefaultDMQId () throws ConnectException, AdminException
String getDefaultDMQId (int serverId) throws ConnectException, AdminException
void setDefaultDMQId (String dmqid) throws ConnectException, AdminException
void setDefaultDMQId (int serverId, String dmqid) throws ConnectException, AdminException
Queue getDefaultDMQ () throws ConnectException, AdminException
  
```

```

Queue getDefaultDMQ (int serverId) throws ConnectException, AdminException
void setDefaultDMQ (Queue dmq) throws ConnectException, AdminException
void setDefaultDMQ (int serverId, Queue dmq) throws ConnectException, AdminException
int getDefaultThreshold () throws ConnectException, AdminException
int getDefaultThreshold (int serverId) throws ConnectException, AdminException
void setDefaultThreshold (int threshold) throws ConnectException, AdminException
void setDefaultThreshold (int serverId, int threshold) throws ConnectException, AdminException
int[] getServerIds () throws ConnectException, AdminException
int[] getServerIds (String domain) throws ConnectException, AdminException
String[] getServerNames () throws ConnectException, AdminException
String[] getServerNames (String domain) throws ConnectException, AdminException
Server[] getServers () throws ConnectException, AdminException
Server[] getServers (String domain) throws ConnectException, AdminException
String[] getDomainNames (int serverId) throws ConnectException, AdminException
Destination[] getDestinations () throws ConnectException, AdminException
Destination[] getDestinations (int serverId) throws ConnectException, AdminException
Destination createQueue (String name) throws AdminException, ConnectException
Destination createQueue (int serverId, String name) throws AdminException, ConnectException
Destination createQueue (int serverId, String name, String className, Properties prop) throws ConnectException,
AdminException
Destination createTopic (String name) throws AdminException, ConnectException
Destination createTopic (int serverId, String name) throws AdminException, ConnectException
Destination createTopic (int serverId, String name, String className, Properties prop) throws ConnectException,
AdminException
User[] getUsers () throws ConnectException, AdminException
User[] getUsers (int serverId) throws ConnectException, AdminException
User createUser (String name, String password) throws ConnectException, AdminException
User createUser (String name, String password, int serverId) throws ConnectException, AdminException
User createUser (String name, String password, String identityClass) throws AdminException, ConnectException
User createUser (String name, String password, int serverId, String identityClassName) throws ConnectException,
AdminException
User createUser (String name, String password, int serverId, String identityClassName, Properties prop) throws
ConnectException, AdminException
Server getLocalServer () throws ConnectException, AdminException
int getLocalServerId () throws ConnectException, AdminException
String getLocalHost () throws ConnectException, AdminException
String getLocalName () throws ConnectException, AdminException
String invokeStaticServerMethod (int serverId, String className, String methodName, Class<?>[] parameterTypes,
Object[] args) throws ConnectException, AdminException
String addAMQPBridgeConnection (int serverId, String urls) throws ConnectException, AdminException
String deleteAMQPBridgeConnection (int serverId, String names) throws ConnectException, AdminException
String addJMSPBridgeConnection (int serverId, String urls) throws ConnectException, AdminException
String deleteJMSPBridgeConnection (int serverId, String names) throws ConnectException, AdminException

```

Detailed Description

The AdminItf interface defines the set of methods needed for administration and monitoring of the Joram platform.

Member Function Documentation

String AdminItf.addAMQPBridgeConnection (int serverId, String urls) throws ConnectException, AdminException

Adds an AMQP server and starts a live connection with it, accessible via the url provided. A server is uniquely identified by the given name. Adding an existing server won't do anything.

Parameters:

serverId	the serverId
urls	the amqp url list identifying the servers separate by space, for example:

	amqp://user:pass@localhost:5672/?name=serv1 amqp://user:pass@localhost: 5678/?name=serv2
--	---

Returns:

the result of the method

Exceptions:

ConnectException	If the connection fails.
AdminException	If the invocation can't be done or fails

void AdminIf.addDomain (String domain, int sid, int port) throws ConnectException, AdminException

Adds a domain to the platform.

The domain will use the default network component "SimpleNetwork".

Parameters:

domain	Name of the added domain.
sid	Id of the router server that gives access to the added domain.
port	Listening port in the added domain of the router server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void AdminIf.addDomain (String domain, String network, int sid, int port) throws ConnectException, AdminException

Adds a domain to the platform using a specific network component.

Parameters:

domain	Name of the added domain.
network	Classname of the network component to use.
sid	Id of the router server that gives access to the added domain.
port	Listening port in the added domain of the router server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

String AdminIf.addJMSBridgeConnection (int serverId, String urls) throws ConnectException, AdminException

Adds a JMS server and starts a live connection with it, accessible via the url provided. A server is uniquely identified by the given name. Adding an existing server won't do anything.

Parameters:

serverId	the serverId
urls	the jms url list identifying the servers separate by space. ex: jndi_url/?name=cnx1&cf=cfName&jndiFactoryClass=com.xxx.yyy&user=user1&pass=pass1&clientID=clientID

Returns:

the result of the method

Exceptions:

ConnectException	If the connection fails.
AdminException	If the invocation can't be done or fails

void AdminItf.addServer (int sid, String host, String domain, int port, String server) throws ConnectException, AdminException

Adds a server to the platform.

The server is configured without any service.

Parameters:

sid	Id of the added server
host	Address of the host where the added server is started
domain	Name of the domain where the server is added
port	Listening port of the server in the specified domain
server	Name of the added server

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

See also:

addServer(int, String, String, int, String, String[], String[])

void AdminItf.addServer (int sid, String host, String domain, int port, String server, String[] services, String[] args) throws ConnectException, AdminException

Adds a server to the platform.

Parameters:

sid	Id of the added server
host	Address of the host where the added server is started
domain	Name of the domain where the server is added
port	Listening port of the server in the specified domain
server	Name of the added server
services	Names of the service to start within the server
args	Services' arguments

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void AdminItf.close ()

Closes the underlying requestor.

Destination AdminItf.createQueue (String name) throws AdminException, ConnectException

Creates or retrieves a queue destination on a given JORAM server.

Parameters:

name	The name of the queue.
------	------------------------

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

See also:

createQueue(int, String, String, Properties)

Destination AdminItf.createQueue (int serverId, String name) throws AdminException, ConnectException

Creates or retrieves a queue destination on a given JORAM server.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the queue.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

See also:

createQueue(int, String, String, Properties)

Destination AdminIf.createQueue (int serverId, String name, String className, Properties prop) throws ConnectException, AdminException

Creates or retrieves a queue destination on a given JORAM server.

First a destination with the specified name is searched on the given server, if it does not exist it is created.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the queue.
className	The queue class name.
prop	The queue properties.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

Destination AdminIf.createTopic (String name) throws AdminException, ConnectException

Creates or retrieves a topic destination on the underlying JORAM server.

Parameters:

name	The name of the topic.
------	------------------------

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

See also:

createTopic(int, String, String, Properties)

Destination AdminIf.createTopic (int serverId, String name) throws AdminException, ConnectException

Creates or retrieves a topic destination on a given JORAM server.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the topic.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

See also:

createTopic(int, String, String, Properties)

Destination AdminItf.createTopic (int serverId, String name, String className, Properties prop) throws ConnectException, AdminException

Creates or retrieves a topic destination on a given JORAM server.

First a destination with the specified name is searched on the given server, if it does not exist it is created.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the topic.
className	The topic class name.
prop	The topic properties.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

User AdminItf.createUser (String name, String password) throws ConnectException, AdminException

Creates or retrieves a user on the underlying JORAM server.

Parameters:

name	Name of the user.
password	Password of the user.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

See also:

[createUser\(String, String, int, String\)](#)

User AdminItf.createUser (String name, String password, int serverId) throws ConnectException, AdminException

Creates or retrieves a user on the underlying JORAM server.

Parameters:

name	Name of the user.
password	Password of the user.
serverId	The identifier of the user's server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

See also:

[createUser\(String, String, int, String\)](#)

User AdminItf.createUser (String name, String password, String identityClass) throws AdminException, ConnectException

Creates or retrieves a user on the underlying JORAM server.

Parameters:

name	Name of the user.
password	Password of the user.
identityClass	Classname for authentication, by default SimpleIdentity for

	user/password.
--	----------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

See also:

createUser(String, String, int, String)

User AdminIf.createUser (String name, String password, int serverId, String identityClassName) throws ConnectException, AdminException

Admin method creating a user for a given server and instantiating the corresponding User object. If the user has already been set on this server, the method simply returns the corresponding User object. Its fails if the target server does not belong to the platform, or if a proxy could not be deployed server side for a new user.

Parameters:

name	Name of the user.
password	Password of the user.
serverId	The identifier of the user's server.
identityClassName	By default user/password for SimpleIdentity.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

User AdminIf.createUser (String name, String password, int serverId, String identityClassName, Properties prop) throws ConnectException, AdminException

Admin method creating a user for a given server and instantiating the corresponding User object. If the user has already been set on this server, the method simply returns the corresponding User object. Its fails if the target server does not belong to the platform, or if a proxy could not be deployed server side for a new user.

Parameters:

name	Name of the user.
password	Password of the user.
serverId	The identifier of the user's server.
identityClassName	By default user/password for SimpleIdentity.
prop	properties

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

String AdminIf.deleteAMQPBridgeConnection (int serverId, String names) throws ConnectException, AdminException

Removes the live connection to the specified AMQP server.

Parameters:

serverId	the serverId
names	the name identifying the server or list of name separate by space

Returns:

the result of the method

Exceptions:

ConnectException	If the connection fails.
------------------	--------------------------

n	
AdminException	If the invocation can't be done or fails

String AdminItf.deleteJMSPBridgeConnection (int serverId, String names) throws ConnectException, AdminException

Removes the live connection to the specified AMQP server.

Parameters:

serverId	the serverId
names	the name identifying the server or list of name separate by space

Returns:

the result of the method

Exceptions:

ConnectException	If the connection fails.
AdminException	If the invocation can't be done or fails

String AdminItf.getConfiguration () throws ConnectException, AdminException

Returns the current servers configuration (a3servers.xml).

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

Queue AdminItf.getDefaultDMQ () throws ConnectException, AdminException

Returns the default dead message queue for the local server, null if not set.

Returns:

The default dead message queue for the local server, null if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

getDefaultDMQ(int)

Queue AdminItf.getDefaultDMQ (int serverId) throws ConnectException, AdminException

Returns the default dead message queue for a given server, null if not set.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Unique identifier of the server.
----------	----------------------------------

Returns:

The default dead message queue for the local server, null if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

String AdminItf.getDefaultDMQId () throws ConnectException, AdminException

Returns the unique identifier of the default dead message queue for the local server, null if not set.

Returns:

The unique identifier of the default dead message queue for the local server, null if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

getDefaultDMQId(int)

String AdminItf.getDefaultDMQId (int serverId) throws ConnectException, AdminException

Returns the unique identifier of the default dead message queue for a given server, null if not set.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Unique identifier of the server.
----------	----------------------------------

Returns:

The unique identifier of the default dead message queue for the local server, null if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

int AdminItf.getDefaultThreshold () throws ConnectException, AdminException

Returns the default threshold value for the local server, -1 if not set.

Returns:

The default threshold value for the local server, -1 if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

getDefaultThreshold(int)

int AdminItf.getDefaultThreshold (int serverId) throws ConnectException, AdminException

Returns the default threshold value for a given server, -1 if not set.

The request fails if the target server does not belong to the platform.

Returns:

The default threshold value for the local server, -1 if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

Destination [] AdminItf.getDestinations () throws ConnectException, AdminException

Returns the list of all destinations that exist on the local server.

Returns:

An array containing all destinations defined on the given server or null if none exists.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	Never thrown.

See also:

getDestinations(int)

Destination [] AdminIf.getDestinations (int serverId) throws ConnectException, AdminException

Returns the list of all destinations that exist on a given server.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Unique identifier of the server.
----------	----------------------------------

Returns:

An array containing all destinations defined on the given server or null if none exists.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

String [] AdminIf.getDomainNames (int serverId) throws ConnectException, AdminException

Returns the list of the domain names that contains the specified server.

Parameters:

serverId	Unique identifier of the server.
----------	----------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

String AdminIf.getLocalHost () throws ConnectException, AdminException

Returns the host name of the server the module is connected to.

Exceptions:

ConnectException	If the admin connection is not established.
AdminException	If the request fails.

See also:

getLocalServer()

String AdminIf.getLocalName () throws ConnectException, AdminException

Returns the name of the server the module is connected to.

Exceptions:

ConnectException	If the admin connection is not established.
AdminException	If the request fails.

See also:

getLocalServer()

Server AdminItf.getLocalServer () throws ConnectException, AdminException

Returns the information about the current server: unique identifier, symbolic name and hostname.

Returns:

The description of the server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

int AdminItf.getLocalServerId () throws ConnectException, AdminException

Returns the identifier of the server the module is connected to.

Exceptions:

ConnectException	If the admin connection is not established.
AdminException	If the request fails.

See also:

getLocalServer()

Server [] AdminItf.getServers () throws ConnectException, AdminException

Returns the list of the platform's servers' identifiers.

Returns:

An array containing the description of all servers.

Exceptions:

ConnectException	
AdminException	

See also:

getServers(String)

Server [] AdminItf.getServers (String domain) throws ConnectException, AdminException

Returns the list of the servers' that belong to the specified domain.

Parameters:

domain	Name of the domain.
--------	---------------------

Returns:

An array containing the description of all servers.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

int [] AdminItf.getServerIds () throws ConnectException, AdminException

Returns the list of the platform's servers' identifiers.

Returns:

An array containing the list of server's identifiers.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.



Copyright ScalAgent D.T. 2023 – All rights reserved

219

See also:

`getServers(String)`

`int [] AdminItf.getServersIds (String domain) throws ConnectException, AdminException`

Returns the list of the servers' identifiers that belong to the specified domain

Parameters:

<code>domain</code>	Name of the domain.
---------------------	---------------------

Returns:

An array containing the list of server's identifiers of the specified domain.

Exceptions:

<code>ConnectException</code>	If the connection fails.
<code>AdminException</code>	Never thrown.

`String [] AdminItf.getServersNames () throws ConnectException, AdminException`

Returns the list of the platform's servers' names.

Returns:

An array containing the list of server's names.

Exceptions:

<code>ConnectException</code>	If the connection fails.
<code>AdminException</code>	Never thrown.

See also:

`getServers(String)`

`String [] AdminItf.getServersNames (String domain) throws ConnectException, AdminException`

Returns the list of the servers' names that belong to the specified domain

Parameters:

<code>domain</code>	Name of the domain.
---------------------	---------------------

Returns:

An array containing the list of server's names of the specified domain.

Exceptions:

<code>ConnectException</code>	If the connection fails.
<code>AdminException</code>	Never thrown.

`Hashtable AdminItf.getStatistics () throws ConnectException, AdminException`

Returns statistics for the local server.

Returns:

statistics for the local server.

Exceptions:

<code>ConnectException</code>	If the connection fails.
<code>AdminException</code>	Never thrown.

See also:

`getStatistics(int)`

Hashtable AdminItf.getStatistics (int serverId) throws ConnectException, AdminException

Returns statistics for the the specified server.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Unique identifier of the server.
----------	----------------------------------

Returns:

the statistics for the the specified server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

long AdminItf.getTimeOutToAbortRequest () throws ConnectException

Returns the maximum time in ms before aborting request.

Returns:

the maximum time in ms before aborting request.

Exceptions:

ConnectException	if the connection is not established.
------------------	---------------------------------------

User [] AdminItf.getUsers () throws ConnectException, AdminException

Returns the list of all users that exist on the local server.

Returns:

An array containing all users defined on the local server, or null if none exist.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

getUsers(int)

User [] AdminItf.getUsers (int serverId) throws ConnectException, AdminException

Returns the list of all users that exist on a given server.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Unique identifier of the given server.
----------	--

Returns:

An array containing all users defined on the local server, or null if none exist.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

String AdminItf.invokeStaticServerMethod (int serverId, String className, String methodName, Class<?>[] parameterTypes, Object[] args) throws ConnectException, AdminException

Invokes the specified static method with the specified parameters on the chosen server. The parameters types of the invoked method must be java primitive types, the java objects wrapping them or String type.

Parameters:

serverId	the identifier of the server.
className	the name of the class holding the static method
methodName	the name of the invoked method
parameterTypes	the list of parameters
args	the arguments used for the method call

Returns:

the result of the invoked method after applying Object#toString() method

Exceptions:

ConnectException	If the connection fails.
AdminException	If the invocation can't be done or fails

boolean AdminIf.isClosed ()

Returns true if the underlying requestor is closed.

Returns:

true if the underlying requestor is closed.

void AdminIf.removeDomain (String domain) throws ConnectException, AdminException

Removes a domain from the platform.

Parameters:

domain	Name of the domain to remove
--------	------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void AdminIf.removeServer (int sid) throws ConnectException, AdminException

Removes a server from the platform.

Parameters:

sid	Id of the removed server
-----	--------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void AdminIf.setDefaultDMQ (Queue dmq) throws ConnectException, AdminException

Sets a given dead message queue as the default DMQ for the local server (null for unsetting previous DMQ).

Parameters:

dmq	The dmq to be set as the default one.
-----	---------------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

setDefaultDMQ(int, Queue)

```
void AdminItf.setDefaultDMQ (int serverId, Queue dmq) throws ConnectException, AdminException
```

Sets a given dead message queue as the default DMQ for a given server (null for unsetting previous DMQ). The request fails if the target server does not belong to the platform.

Parameters:

serverId	The identifier of the server.
dmq	The dmq to be set as the default one.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

```
void AdminItf.setDefaultDMQId (String dmqid) throws ConnectException, AdminException
```

Sets a given dead message queue as the default DMQ for the local server (null for unsetting previous DMQ).

Parameters:

dmqid	The dmqid (AgentId) to be set as the default one.
-------	---

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

`setDefaultDMQId(int, String)`

```
void AdminItf.setDefaultDMQId (int serverId, String dmqid) throws ConnectException, AdminException
```

Sets a given dead message queue as the default DMQ for a given server (null for unsetting previous DMQ). The request fails if the target server does not belong to the platform.

Parameters:

serverId	The identifier of the server.
dmqid	The dmqid (AgentId) to be set as the default one.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

```
void AdminItf.setDefaultThreshold (int threshold) throws ConnectException, AdminException
```

Sets a given value as the default threshold for the local server (-1 for unsetting previous value).

Parameters:

threshold	The threshold value to be set.
-----------	--------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

`setDefaultThreshold(int, int)`

void AdminIf.setDefaultThreshold (int serverId, int threshold) throws ConnectException, AdminException

Sets a given value as the default threshold for a given server (-1 for unsetting previous value). The request fails if the target server does not belong to the platform.

Parameters:

serverId	The identifier of the server.
threshold	The threshold value to be set.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void AdminIf.setTimeOutToAbortRequest (long timeOut) throws ConnectException

Set the maximum time in ms before aborting request.

Parameters:

timeOut	the maximum time in ms before aborting request.
---------	---

Exceptions:

ConnectException	if the connection is not established.
------------------	---------------------------------------

void AdminIf.stopServer () throws ConnectException, AdminException

Stops the platform local server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

See also:

stopServer(int)

void AdminIf.stopServer (int serverId) throws ConnectException, AdminException

Stops a given server of the platform.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Identifier of the server to stop.
----------	-----------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

11.4. JoramSaxWrapper

Class org.objectweb.joram.client.jms.admin.JoramSaxWrapper

Public Member Functions

JoramSaxWrapper ()
JoramSaxWrapper (AdminWrapper defaultWrapper)
void parse (Reader cfgReader, String cfgName) throws Exception

Detailed Description

XML SAX Wrapper for Joram Administration configuration file.

Constructor Documentation

JoramSaxWrapper.JoramSaxWrapper ()

Builds a new JoramSaxWrapper using by default AdminModule static connection.

JoramSaxWrapper.JoramSaxWrapper (AdminWrapper defaultWrapper)

Builds a new JoramSaxWrapper using by default the given administration connection.

Parameters:

defaultWrapper	The administration connection to use by default.
----------------	--

Member Function Documentation

void JoramSaxWrapper.parse (Reader cfgReader, String cfgName) throws Exception

Launches the XML parser.

11.5. AdminModule

Class org.objectweb.joram.client.jms.admin.AdminModule

Static Public Member Functions

```
static AdminWrapper getWrapper () throws ConnectException
static void main (String[] args)
static void connect (javax.jms.ConnectionFactory cf) throws ConnectException, AdminException
static void connect (javax.jms.ConnectionFactory cf, String name, String password) throws ConnectException, AdminException
static void connect (javax.jms.ConnectionFactory cf, String name, String password, String identityClass) throws ConnectException, AdminException
static void connect (javax.jms.TopicConnectionFactory cf, String name, String password) throws ConnectException, AdminException
static void connect (javax.jms.TopicConnectionFactory cf, String name, String password, String identityClass) throws ConnectException, AdminException
static void connect () throws UnknownHostException, ConnectException, AdminException
static void connect (String name, String password) throws UnknownHostException, ConnectException, AdminException
static void connect (String name, String password, int cnxTimer) throws UnknownHostException, ConnectException, AdminException
static void connect (String host, int port, String name, String password) throws UnknownHostException, ConnectException, AdminException
static void connect (String host, int port, String name, String password, int cnxTimer) throws UnknownHostException, ConnectException, AdminException
static void connect (String name, String password, int cnxTimer, String reliableClass) throws UnknownHostException, ConnectException, AdminException
static void connect (String host, int port, String name, String password, int cnxTimer, String reliableClass) throws UnknownHostException, ConnectException, AdminException
static void connect (String host, int port, String name, String password, int cnxTimer, String reliableClass, String identityClass) throws UnknownHostException, ConnectException, AdminException
static void collocatedConnect () throws ConnectException, AdminException
```

```

static void collocatedConnect (String name, String password) throws ConnectException, AdminException
static void collocatedConnect (String name, String password, String identityClass) throws ConnectException, AdminException
static void doCollocatedConnect (String name, String password, String identityClass) throws ConnectException, AdminException
static void disconnect ()
static void stopServer (int serverId) throws ConnectException, AdminException
static void stopServer () throws ConnectException, AdminException
static void addServer (int sid, String host, String domain, int port, String server) throws ConnectException, AdminException
static void addServer (int sid, String host, String domain, int port, String server, String[] services, String[] args) throws ConnectException, AdminException
static void removeServer (int sid) throws ConnectException, AdminException
static void addDomain (String domain, int sid, int port) throws ConnectException, AdminException
static void addDomain (String domain, String network, int sid, int port) throws ConnectException, AdminException
static void removeDomain (String domain) throws ConnectException, AdminException
static String getConfiguration () throws ConnectException, AdminException
static Hashtable getStatistics () throws ConnectException, AdminException
static Hashtable getStatistics (int serverId) throws ConnectException, AdminException
static Queue getDefaultDMQ () throws ConnectException, AdminException
static Queue getDefaultDMQ (int serverId) throws ConnectException, AdminException
static void setDefaultDMQ (Queue dmq) throws ConnectException, AdminException
static void setDefaultDMQ (int serverId, Queue dmq) throws ConnectException, AdminException
static String getDefaultDMQId () throws ConnectException, AdminException
static String getDefaultDMQId (int serverId) throws ConnectException, AdminException
static void setDefaultDMQId (String dmqlId) throws ConnectException, AdminException
static void setDefaultDMQId (int serverId, String dmqlId) throws ConnectException, AdminException
static int getDefaultThreshold () throws ConnectException, AdminException
static int getDefaultThreshold (int serverId) throws ConnectException, AdminException
static void setDefaultThreshold (int threshold) throws ConnectException, AdminException
static void setDefaultThreshold (int serverId, int threshold) throws ConnectException, AdminException
static List getServersIds () throws ConnectException, AdminException
static List getServersIds (String domain) throws ConnectException, AdminException
static Server[] getServers (String domain) throws ConnectException, AdminException
static String[] getDomainNames (int serverId) throws ConnectException, AdminException
static List getDestinationsList () throws ConnectException, AdminException
static List getDestinationsList (int serverId) throws ConnectException, AdminException
static Destination[] getDestinations () throws ConnectException, AdminException
static Destination[] getDestinations (int serverId) throws ConnectException, AdminException
static Destination createQueue (int serverId, String name, String className, Properties prop) throws ConnectException, AdminException
static Destination createTopic (int serverId, String name, String className, Properties prop) throws ConnectException, AdminException
static List getUsersList () throws ConnectException, AdminException
static List getUsersList (int serverId) throws ConnectException, AdminException
static User[] getUsers () throws ConnectException, AdminException
static User[] getUsers (int serverId) throws ConnectException, AdminException
static User createUser (String name, String password, int serverId, String identityClassName) throws ConnectException, AdminException
static int getLocalServerId () throws ConnectException, AdminException
static String getLocalHost () throws ConnectException
static int getLocalPort () throws ConnectException
static void executeXMLAdmin (String cfgDir, String cfgFileName) throws Exception
static void executeXMLAdmin (String path) throws Exception
static void exportRepositoryToFile (String exportDir, String exportFilename) throws AdminException
static void setTimeOutToAbortRequest (long timeOut) throws ConnectException
static long getTimeOutToAbortRequest () throws ConnectException
static String invokeStaticServerMethod (String className, String methodName, Class<?>[] parameterTypes, Object[] args) throws ConnectException, AdminException
static String invokeStaticServerMethod (int serverId, String className, String methodName, Class<?>[] parameterTypes, Object[] args) throws ConnectException, AdminException

```

```
static String addAMQPBridgeConnection (int serverId, String urls) throws ConnectException, AdminException
static String deleteAMQPBridgeConnection (int serverId, String names) throws ConnectException, AdminException
static String addJMSBridgeConnection (int serverId, String urls) throws ConnectException, AdminException
static String deleteJMSPBridgeConnection (int serverId, String names) throws ConnectException, AdminException
```

Detailed Description

The AdminModule class allows to set an administrator connection to a given JORAM server, and provides administration and monitoring methods at a server/platform level.

The AdminModule class uses a unique static connection to the Joram server, the connection is opened through connect method and closed by calling disconnect.

See also:

[AdminWrapper](#)

Member Function Documentation

static String AdminModule.addAMQPBridgeConnection (int serverId, String urls) throws ConnectException, AdminException [static]

Adds an AMQP server and starts a live connection with it, accessible via the url provided. A server is uniquely identified by the given name. Adding an existing server won't do anything.

Parameters:

serverId	the serverId
urls	the amqp url list identifying the servers separate by space, for example: amqp://user:pass@localhost:5672/?name=serv1 amqp://user:pass@localhost: 5678/?name=serv2

Returns:

the result of the method

Exceptions:

ConnectException	If the connection fails.
AdminException	If the invocation can't be done or fails

static void AdminModule.addDomain (String domain, int sid, int port) throws ConnectException, AdminException [static]

Adds a domain to the platform.

Parameters:

domain	Name of the added domain.
sid	Id of the router server that gives access to the added domain.
port	Listening port in the added domain of the router server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

static void AdminModule.addDomain (String domain, String network, int sid, int port) throws ConnectException, AdminException [static]

Adds a domain to the platform using a specific network component.

Parameters:

domain	Name of the added domain.
network	Classname of the network component to use.

sid	Id of the router server that gives access to the added domain.
port	Listening port in the added domain of the router server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

static String AdminModule.addJMSBridgeConnection (int serverId, String urls) throws ConnectException, AdminException [static]

Adds a JMS server and starts a live connection with it, accessible via the url provided. A server is uniquely identified by the given name. Adding an existing server won't do anything.

Parameters:

serverId	the serverId
urls	the jms url list identifying the servers separate by space. ex: jndi_url/?name=cnx1&cf=cfName&jndiFactoryClass=com.xxx.yyy&user=user1&pass=pass1&clientID=clientID

Returns:

the result of the method

Exceptions:

ConnectException	If the connection fails.
AdminException	If the invocation can't be done or fails

static void AdminModule.addServer (int sid, String host, String domain, int port, String server) throws ConnectException, AdminException [static]

Adds a server to the platform.

Parameters:

sid	Id of the added server
host	Address of the host where the added server is started
domain	Name of the domain where the server is added
port	Listening port of the server in the specified domain
server	Name of the added server

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

static void AdminModule.addServer (int sid, String host, String domain, int port, String server, String[] services, String[] args) throws ConnectException, AdminException [static]

Adds a server to the platform.

Parameters:

sid	Id of the added server
host	Address of the host where the added server is started
domain	Name of the domain where the server is added
port	Listening port of the server in the specified domain
server	Name of the added server
services	Names of the service to start within the server
args	Services' arguments

Exceptions:

ConnectException	If the connection fails.

AdminException	If the request fails.
----------------	-----------------------

static void AdminModule.collocatedConnect () throws ConnectException, AdminException [static]

Opens a connection with the colocated Joram server.

Default administrator login name and password are used for connection as defined in AbstractConnectionFactory#getDefaultRootLogin() and AbstractConnectionFactory#getDefaultRootPassword().

Exceptions:

UnknownHostException	Never thrown.
ConnectException	If connecting fails.
AdminException	If the administrator identification is incorrect.

static void AdminModule.collocatedConnect (String name, String password) throws ConnectException, AdminException [static]

Opens a connection with the colocated JORAM server.

Parameters:

name	Administrator's name.
password	Administrator's password.

Exceptions:

ConnectException	If connecting fails.
AdminException	If the administrator identification is incorrect.

void AdminModule.collocatedConnect (String name, String password, String identityClass) throws ConnectException, AdminException [static]

Opens a connection with the colocated JORAM server.

Parameters:

name	Administrator's name.
password	Administrator's password.
identityClass	identity class name.

Exceptions:

ConnectException	If connecting fails.
AdminException	If the administrator identification is incorrect.

Deprecated:

Next to Joram 5.2 use connect methods with ConnectionFactory.

static void AdminModule.connect (javax.jms.ConnectionFactory cf) throws ConnectException, AdminException [static]

Opens a connection dedicated to administering with the Joram server which parameters are wrapped by a given ConnectionFactory . Default administrator login name and password are used for connection as defined in AbstractConnectionFactory#getDefaultRootLogin() and AbstractConnectionFactory#getDefaultRootPassword().

Parameters:

cf	The Joram's ConnectionFactory to use for connecting.
----	--

Exceptions:

ConnectException	If connecting fails.
AdminException	If the administrator identification is incorrect.
ClassCastException	If the ConnectionFactory is not a Joram ConnectionFactory.

static void AdminModule.connect (javax.jms.ConnectionFactory cf, String name, String password) throws ConnectException, AdminException [static]

Opens a connection dedicated to administering with the Joram server which parameters are wrapped by a given ConnectionFactory .

Parameters:

cf	The Joram's ConnectionFactory to use for connecting.
name	Administrator's name.
password	Administrator's password.

Exceptions:

ConnectException	If connecting fails.
AdminException	If the administrator identification is incorrect.
ClassCastException	If the ConnectionFactory is not a Joram ConnectionFactory.

static void AdminModule.connect (javax.jms.ConnectionFactory cf, String name, String password, String identityClass) throws ConnectException, AdminException [static]

Opens a connection dedicated to administering with the Joram server which parameters are wrapped by a given ConnectionFactory .

Parameters:

cf	The Joram's ConnectionFactory to use for connecting.
name	Administrator's name.
password	Administrator's password.
identityClass	identity class name.

Exceptions:

ConnectException	If connecting fails.
AdminException	If the administrator identification is incorrect.
ClassCastException	If the ConnectionFactory is not a Joram ConnectionFactory.

static void AdminModule.connect (javax.jms.TopicConnectionFactory cf, String name, String password) throws ConnectException, AdminException [static]

Opens a connection dedicated to administering with the Joram server which parameters are wrapped by a given TopicConnectionFactory .

Parameters:

cf	The TopicConnectionFactory to use for connecting.
name	Administrator's name.
password	Administrator's password.

Exceptions:

ConnectException	If connecting fails.
AdminException	If the administrator identification is incorrect.
ClassCastException	If the ConnectionFactory is not a Joram ConnectionFactory.

Deprecated:

No longer use TopicConnectionFactory next to Joram 5.2.

static AdminModule.connect (javax.jms.TopicConnectionFactory cf, String name, String password, String identityClass) throws ConnectException, AdminException [static]

Opens a connection dedicated to administering with the Joram server which parameters are wrapped by a given TopicConnectionFactory .

Parameters:

cf	The TopicConnectionFactory to use for connecting.
name	Administrator's name.
password	Administrator's password.
identityClass	identity class name.

Exceptions:

ConnectException	If connecting fails.
AdminException	If the administrator identification is incorrect.
ClassCastException	If the ConnectionFactory is not a Joram ConnectionFactory.

Deprecated:

No longer use TopicConnectionFactory next to Joram 5.2.

static void AdminModule.connect () throws UnknownHostException, ConnectException, AdminException [static]

Opens a TCP connection with the Joram server running on the default "localhost" host and listening to the default 16010 port. Default administrator login name and password are used for connection as defined in AbstractConnectionFactory#getDefaultRootLogin() and AbstractConnectionFactory#getDefaultRootPassword().

Exceptions:

UnknownHostException	Never thrown.
ConnectException	If connecting fails.
AdminException	If the administrator identification is incorrect.

static void AdminModule.connect (String name, String password) throws UnknownHostException, ConnectException, AdminException [static]

Opens a TCP connection with the Joram server running on the default "localhost" host and listening to the default 16010 port.

Parameters:

name	Administrator's name.
password	Administrator's password.

Exceptions:

UnknownHostException	Never thrown.
ConnectException	If connecting fails.
AdminException	If the administrator identification is incorrect.

static void AdminModule.connect (String name, String password, int cnxTimer) throws UnknownHostException, ConnectException, AdminException [static]

Opens a TCP connection with the Joram server running on the default "localhost" host and listening to the default 16010 port.

Parameters:

name	Administrator's name.
password	Administrator's password.
cnxTimer	Timer in seconds during which connecting to the server is attempted.

Exceptions:

UnknownHostException	Never thrown.
ConnectException	If connecting fails.
AdminException	If the administrator identification is incorrect.

Deprecated:

Next to Joram 5.2 use connect methods with ConnectionFactory.

static void AdminModule.connect (String host, int port, String name, String password) throws UnknownHostException, ConnectException, AdminException [static]

Opens a TCP connection with the Joram server running on a given host and listening to a given port.

Parameters:

host	The name or IP address of the host the server is running on.
port	The number of the port the server is listening to.
name	Administrator's name.
password	Administrator's password.

Exceptions:

UnknownHostException	If the host is invalid.
ConnectException	If connecting fails.
AdminException	If the administrator identification is incorrect.

static void AdminModule.connect (String host, int port, String name, String password, int cnxTimer) throws UnknownHostException, ConnectException, AdminException [static]

Opens a TCP connection with the Joram server running on a given host and listening to a given port.

Parameters:

host	The name or IP address of the host the server is running on.
port	The number of the port the server is listening to.
name	Administrator's name.
password	Administrator's password.
cnxTimer	Timer in seconds during which connecting to the server is attempted.

Exceptions:

UnknownHostException	If the host is invalid.
ConnectException	If connecting fails.
AdminException	If the administrator identification is incorrect.

Deprecated:

Next to Joram 5.2 use connect methods with ConnectionFactory.

static void AdminModule.connect (String name, String password, int cnxTimer, String reliableClass) throws UnknownHostException, ConnectException, AdminException [static]

Opens a TCP connection with the Joram server running on the default "localhost" host and listening to the default 16010 port.

Parameters:

name	Administrator's name.
password	Administrator's password.
cnxTimer	Timer in seconds during which connecting to the server is attempted.
reliableClass	Reliable class name.

Exceptions:

UnknownHostException	Never thrown.
ConnectException	If connecting fails.
AdminException	If the administrator identification is incorrect.

Deprecated:

Next to Joram 5.2 use connect methods with ConnectionFactory.

static void AdminModule.connect (String host, int port, String name, String password, int cnxTimer, String reliableClass) throws UnknownHostException, ConnectException, AdminException [static]

Opens a TCP connection with the Joram server running on a given host and listening to a given port.

Parameters:

host	The name or IP address of the host the server is running on.
------	--

port	The number of the port the server is listening to.
name	Administrator's name.
password	Administrator's password.
cnxTimer	Timer in seconds during which connecting to the server is attempted.
reliableClass	Reliable class name.

Exceptions:

UnknownHostException	If the host is invalid.
ConnectException	If connecting fails.
AdminException	If the administrator identification is incorrect.

Deprecated:

Next to Joram 5.2 use connect methods with ConnectionFactory.

```
static void AdminModule.connect (String host, int port, String name, String password, int cnxTimer, String reliableClass, String identityClass) throws UnknownHostException, ConnectException, AdminException [static]
```

Opens a TCP connection with the Joram server running on a given host and listening to a given port.

Parameters:

host	The name or IP address of the host the server is running on.
port	The number of the port the server is listening to.
name	Administrator's name.
password	Administrator's password.
cnxTimer	Timer in seconds during which connecting to the server is attempted.
reliableClass	Reliable class name.
identityClass	identity class name.

Exceptions:

UnknownHostException	If the host is invalid.
ConnectException	If connecting fails.
AdminException	If the administrator identification is incorrect.

Deprecated:

Next to Joram 5.2 use connect methods with ConnectionFactory.

```
static Destination AdminModule.createQueue (int serverId, String name, String className, Properties prop) throws ConnectException, AdminException [static]
```

Creates or retrieves a queue destination on a given JORMAN server.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the queue.
className	The queue class name.
prop	The queue properties.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

See also:

AdminWrapper::createQueue(int, String, String, Properties)

```
static Destination AdminModule.createTopic (int serverId, String name, String className, Properties prop) throws ConnectException, AdminException [static]
```

Creates or retrieves a topic destination on a given JORMAN server.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the topic.
className	The topic class name.
prop	The topic properties.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

See also:

AdminWrapper::createTopic(int, String, String, Properties)

static User AdminModule.createUser (String name, String password, int serverId, String identityClassName) throws ConnectException, AdminException [static]

Admin method creating a user for a given server and instantiating the corresponding User object.

Parameters:

name	Name of the user.
password	Password of the user.
serverId	The identifier of the user's server.
identityClassName	By default user/password for SimpleIdentity.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

See also:

AdminWrapper::createUser(String, String, int, String)

static String AdminModule.deleteAMQPBridgeConnection (int serverId, String names) throws ConnectException, AdminException [static]

Removes the live connection to the specified AMQP server.

Parameters:

serverId	the serverId
names	the name identifying the server or list of name separate by space

Returns:

the result of the method

Exceptions:

ConnectException	If the connection fails.
AdminException	If the invocation can't be done or fails

static String AdminModule.deleteJMSPBridgeConnection (int serverId, String names) throws ConnectException, AdminException [static]

Removes the live connection to the specified AMQP server.

Parameters:

serverId	the serverId
names	the name identifying the server or list of name separate by space

Returns:

the result of the method

Exceptions:

ConnectException	If the connection fails.
AdminException	If the invocation can't be done or fails

static AdminModule.disconnect () [static]

Closes the administration connection.

```
static void AdminModule.doCollocatedConnect (String name, String password, String identityClass) throws ConnectException, AdminException [static]
```

Opens a connection with the collocated JORAM server.

Parameters:

name	Administrator's name.
password	Administrator's password.
identityClass	identity class name.

Exceptions:

ConnectException	If connecting fails.
AdminException	If the administrator identification is incorrect.

```
static void AdminModule.executeXMLAdmin (String cfgDir, String cfgFileName) throws Exception [static]
```

This method execute the XML script file that the location is given in parameter.

Parameters:

cfgDir	The directory containing the file.
cfgFileName	The script filename.

Since:

4.3.10

```
static void AdminModule.executeXMLAdmin (String path) throws Exception [static]
```

This method execute the XML script file that the pathname is given in parameter.

Parameters:

path	The script pathname.
------	----------------------

Since:

4.3.10

```
static void AdminModule.exportRepositoryToFile (String exportDir, String exportFilename) throws AdminException [static]
```

Export the repository content to an XML file

- only the destinations objects are retrieved in this version
- xml script format of the admin objects (joramAdmin.xml)

Parameters:

exportDir	target directory where the export file will be put
exportFilename	filename of the export file

Exceptions:

AdminException	if an error occurs
----------------	--------------------

static String AdminModule.getConfiguration () throws ConnectException, AdminException [static]

Returns the current servers configuration (a3servers.xml).

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

static Queue AdminModule.getDefaultDMQ () throws ConnectException, AdminException [static]

Returns the default dead message queue for the local server, null if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

static Queue AdminModule.getDefaultDMQ (int serverId) throws ConnectException, AdminException [static]

Returns the default dead message queue for a given server, null if not set.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Unique identifier of the server.
----------	----------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

static String AdminModule.getDefaultDMQId () throws ConnectException, AdminException [static]

Returns the default dead message queue for the local server, null if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

static String AdminModule.getDefaultDMQId (int serverId) throws ConnectException, AdminException [static]

Returns the default dead message queue for a given server, null if not set.

The request fails if the target server does not belong to the platform.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

static int AdminModule.getDefaultThreshold () throws ConnectException, AdminException [static]

Returns the default threshold value for the local server, -1 if not set.

Exceptions:

ConnectException	If the connection fails.
------------------	--------------------------

n	
AdminException	Never thrown.

static int AdminModule.getDefaultThreshold (int serverId) throws ConnectException, AdminException [static]

Returns the default threshold value for a given server, -1 if not set.
The request fails if the target server does not belong to the platform.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

static Destination [] AdminModule.getDestinations () throws ConnectException, AdminException [static]

Returns the list of all destinations that exist on the local server, or null if none exist.

Returns:

An array containing the list of all destinations of the local server or null if none exists.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	Never thrown.

static Destination [] AdminModule.getDestinations (int serverId) throws ConnectException, AdminException [static]

Returns the list of all destinations that exist on a given server, or null if none exist.
The request fails if the target server does not belong to the platform.

Parameters:

serverId	The unique identifier of the selected server.
----------	---

Returns:

An array containing the list of all destinations of the local server or null if none exists.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

static List AdminModule.getDestinationsList () throws ConnectException, AdminException [static]

Returns the list of all destinations that exist on the local server, or an empty list if none exist.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	Never thrown.

Deprecated:

No longer supported next to Joram 5.2

static List AdminModule.getDestinationsList (int serverId) throws ConnectException, AdminException [static]

Returns the list of all destinations that exist on a given server, or an empty list if none exist.
The request fails if the target server does not belong to the platform.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

Deprecated:

No longer supported next to Joram 5.2

static String [] AdminModule.getDomainNames (int serverId) throws ConnectException, AdminException [static]

Returns the list of the domain names that contains the specified server.

Parameters:

serverId	Unique identifier of the server.
----------	----------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

static String AdminModule.getLocalHost () throws ConnectException [static]

Returns the host name of the server the module is connected to.

Exceptions:

ConnectException	If the admin connection is not established.
------------------	---

static int AdminModule.getLocalPort () throws ConnectException [static]

Returns the port number of the server the module is connected to.

Exceptions:

ConnectException	If the admin connection is not established.
------------------	---

static int AdminModule.getLocalServerId () throws ConnectException, AdminException [static]

Returns the identifier of the server the module is connected to.

Exceptions:

ConnectException	If the admin connection is not established.
AdminException	

static Server [] AdminModule.getServers (String domain) throws ConnectException, AdminException [static]

Returns the list of the servers' that belong to the specified domain

Parameters:

domain	Name of the domain.
--------	---------------------

Returns:

An array containing the list of the servers of the specified domain.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

static List AdminModule.getServerIds () throws ConnectException, AdminException [static]

Returns the list of the platform's servers' identifiers.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

Deprecated:

No longer supported next to Joram 5.2

static List AdminModule.getServerIds (String domain) throws ConnectException, AdminException [static]

Returns the list of the servers' identifiers that belong to the specified domain

Parameters:

domain	Name of the domain.
--------	---------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

Deprecated:

No longer supported next to Joram 5.2

static Hashtable AdminModule.getStatistics () throws ConnectException, AdminException [static]

Returns statistics for the local server.

Returns:

statistics for the local server.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

getStatistics(int)

static Hashtable AdminModule.getStatistics (int serverId) throws ConnectException, AdminException [static]

Returns statistics for the the specified server.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Unique identifier of the server.
----------	----------------------------------

Returns:

the statistics for the the specified server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

static long AdminModule.getTimeOutToAbortRequest () throws ConnectException [static]

Gets the timeout in ms before abortion of administration requests.

Returns:

the timeout

Exceptions:

ConnectException	If the connection is not established.
------------------	---------------------------------------

Since:

5.2.2

static User [] AdminModule.getUsers () throws ConnectException, AdminException [static]

Returns the list of all users that exist on the local server, or null if none exist.

Returns:

An array containing all users defined on the local server, or null if none exist.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

static User [] AdminModule.getUsers (int serverId) throws ConnectException, AdminException [static]

Returns the list of all users that exist on a given server, or null if none exist.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Unique identifier of the given server.
----------	--

Returns:

An array containing all users defined on the local server, or null if none exist.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

static List AdminModule.getUsersList () throws ConnectException, AdminException [static]

Returns the list of all users that exist on the local server, or an empty list if none exist.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

Deprecated:

No longer supported next to Joram 5.2

static List AdminModule.getUsersList (int serverId) throws ConnectException, AdminException [static]

Returns the list of all users that exist on a given server, or an empty list if none exist.

The request fails if the target server does not belong to the platform.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

Deprecated:

No longer supported next to Joram 5.2

static AdminWrapper AdminModule.getWrapper () throws ConnectException [static]

Returns the administration wrapper.

Returns:

The administration wrapper.

Exceptions:

ConnectException	if no wrapper is defined.
------------------	---------------------------

static String AdminModule.invokeStaticServerMethod (String className, String methodName, Class<?>[] parameterTypes, Object[] args) throws ConnectException, AdminException [static]

Invokes the specified static method with the specified parameters on the local server. The parameters types of the invoked method must be java primitive types, the java objects wrapping them or String type.

Parameters:

className	the name of the class holding the static method
methodName	the name of the invoked method
parameterTypes	the list of parameters
args	the arguments used for the method call

Returns:

the result of the invoked method after applying Object#toString() method

Exceptions:

ConnectException	If the connection fails.
AdminException	If the invocation can't be done or fails

static String AdminModule.invokeStaticServerMethod (int serverId, String className, String methodName, Class<?>[] parameterTypes, Object[] args) throws ConnectException, AdminException [static]

Invokes the specified static method with the specified parameters on the chosen server. The parameters types of the invoked method must be java primitive types, the java objects wrapping them or String type.

Parameters:

serverId	the identifier of the server.
className	the name of the class holding the static method
methodName	the name of the invoked method
parameterTypes	the list of parameters
args	the arguments used for the method call

Returns:

the result of the invoked method after applying Object#toString() method

Exceptions:

ConnectException	If the connection fails.
AdminException	If the invocation can't be done or fails

static void admin.AdminModule.main (String[] args) [static]

This method execute the XML script file that the path is given in parameter.

Since:

4.3.12

static void AdminModule.removeDomain (String domain) throws ConnectException, AdminException [static]

Removes a domain from the platform.

Parameters:

domain	Name of the added domain
--------	--------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

static void AdminModule.removeServer (int sid) throws ConnectException, AdminException [static]

Removes a server from the platform.

Parameters:

sid	Id of the removed server
-----	--------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

static void AdminModule.setDefaultDMQ (Queue dmq) throws ConnectException, AdminException [static]

Sets a given dead message queue as the default DMQ for the local server (null for unsetting previous DMQ).

Parameters:

dmq	The dmq to be set as the default one.
-----	---------------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

static void AdminModule.setDefaultDMQ (int serverId, Queue dmq) throws ConnectException, AdminException [static]

Sets a given dead message queue as the default DMQ for a given server (null for unsetting previous DMQ). The request fails if the target server does not belong to the platform.

Parameters:

serverId	The identifier of the server.
dmq	The dmq to be set as the default one.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

static void AdminModule.setDefaultDMQId (String dmqlId) throws ConnectException, AdminException [static]

Sets a given dead message queue as the default DMQ for the local server (null for unsetting previous DMQ).

Parameters:

dmqlId	The dmqlId (AgentId) to be set as the default one.
--------	--

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

static void AdminModule.setDefaultDMQId (int serverId, String dmqlId) throws ConnectException, AdminException [static]

Sets a given dead message queue as the default DMQ for a given server (null for unsetting previous DMQ). The request fails if the target server does not belong to the platform.

Parameters:

serverId	The identifier of the server.
dmqlId	The dmqlId (AgentId) to be set as the default one.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

static void AdminModule.setDefaultThreshold (int threshold) throws ConnectException, AdminException [static]

Sets a given value as the default threshold for the local server (-1 for unsetting previous value).

Parameters:

threshold	The threshold value to be set.
-----------	--------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

static void AdminModule.setDefaultThreshold (int serverId, int threshold) throws ConnectException, AdminException [static]

Sets a given value as the default threshold for a given server (-1 for unsetting previous value). The request fails if the target server does not belong to the platform.

Parameters:

serverId	The identifier of the server.
threshold	The threshold value to be set.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

static void AdminModule.setTimeOutToAbortRequest (long timeOut) throws ConnectException [static]

Sets the timeout in ms before abortion of administration requests.

Be careful, the value can be changed prior to the connection only using the AdminRequestor.REQUEST_TIMEOUT_PROP property.

Parameters:

timeOut	The timeout
---------	-------------

Exceptions:

ConnectException	if the connection is not established.
------------------	---------------------------------------

Since:

5.2.2

static void AdminModule.stopServer (int serverId) throws ConnectException, AdminException [static]

Stops a given server of the platform.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Identifier of the server to stop.
----------	-----------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

static void AdminModule.stopServer () throws ConnectException, AdminException [static]

Stops the platform local server.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

11.6. AdminWrapper

Class org.objectweb.joram.client.jms.admin.AdminWrapper

Implements AdminIf

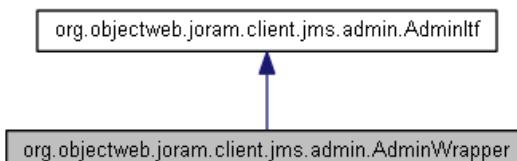


Figure 11.2 - Inheritance diagram for AdminWrapper class

Public Member Functions

```

final void setTimeOutToAbortRequest (long timeOut) throws ConnectException
final long getTimeOutToAbortRequest () throws ConnectException
AdminWrapper (Connection cnx) throws JMSEException, ConnectException, AdminException
void close ()
boolean isClosed ()
final void stopServer () throws ConnectException, AdminException
final void stopServer (int serverId) throws ConnectException, AdminException
final void addServer (int sid, String host, String domain, int port, String server) throws ConnectException, AdminException
final void addServer (int sid, String host, String domain, int port, String server, String[] services, String[] args) throws ConnectException, AdminException
final void removeServer (int sid) throws ConnectException, AdminException
final void addDomain (String domain, int sid, int port) throws ConnectException, AdminException
final void addDomain (String domain, String network, int sid, int port) throws ConnectException, AdminException
final void removeDomain (String domain) throws ConnectException, AdminException
final String getConfiguration () throws ConnectException, AdminException
final Hashtable getStatistics () throws ConnectException, AdminException
  
```

```

final Hashtable getStatistics (int serverId) throws ConnectException, AdminException
final String getDefaultDMQId () throws ConnectException, AdminException
final String getDefaultDMQId (int serverId) throws ConnectException, AdminException
final void setDefaultDMQId (String dmqId) throws ConnectException, AdminException
final void setDefaultDMQId (int serverId, String dmqId) throws ConnectException, AdminException
final Queue getDefaultDMQ () throws ConnectException, AdminException
final Queue getDefaultDMQ (int serverId) throws ConnectException, AdminException
final void setDefaultDMQ (Queue dmq) throws ConnectException, AdminException
final void setDefaultDMQ (int serverId, Queue dmq) throws ConnectException, AdminException
final int getDefaultThreshold () throws ConnectException, AdminException
final int getDefaultThreshold (int serverId) throws ConnectException, AdminException
final void setDefaultThreshold (int threshold) throws ConnectException, AdminException
final void setDefaultThreshold (int serverId, int threshold) throws ConnectException, AdminException
final int[] getServerIds () throws ConnectException, AdminException
final int[] getServerIds (String domain) throws ConnectException, AdminException
final String[] getServerNames () throws ConnectException, AdminException
final String[] getServerNames (String domain) throws ConnectException, AdminException
final Server[] getServers () throws ConnectException, AdminException
final Server[] getServers (String domain) throws ConnectException, AdminException
final String[] getDomainNames (int serverId) throws ConnectException, AdminException
final Destination[] getDestinations () throws ConnectException, AdminException
final Destination[] getDestinations (int serverId) throws ConnectException, AdminException
Destination createQueue (String name) throws AdminException, ConnectException
Destination createQueue (int serverId, String name) throws AdminException, ConnectException
Destination createQueue (int serverId, String name, String className, Properties prop) throws ConnectException, AdminException
Destination createTopic (String name) throws AdminException, ConnectException
Destination createTopic (int serverId, String name) throws AdminException, ConnectException
Destination createTopic (int serverId, String name, String className, Properties prop) throws ConnectException, AdminException
Queue createDeadMQueue (int serverId, String name) throws ConnectException, AdminException
final User[] getUsers () throws ConnectException, AdminException
final User[] getUsers (int serverId) throws ConnectException, AdminException
User createUser (String name, String password) throws ConnectException, AdminException
User createUser (String name, String password, int serverId) throws ConnectException, AdminException
User createUser (String name, String password, String identityClass) throws AdminException, ConnectException
User createUser (String name, String password, int serverId, String identityClassName) throws ConnectException, AdminException
User createUser (String name, String password, int serverId, String identityClassName, Properties prop) throws ConnectException, AdminException
final Server getLocalServer () throws ConnectException, AdminException
final int getLocalServerId () throws ConnectException, AdminException
final String getLocalHost () throws ConnectException, AdminException
final String getLocalName () throws ConnectException, AdminException
String invokeStaticServerMethod (int serverId, String className, String methodName, Class<?>[] parameterTypes, Object[] args) throws ConnectException, AdminException
String addAMQPBridgeConnection (int serverId, String urls) throws ConnectException, AdminException
String deleteAMQPBridgeConnection (int serverId, String names) throws ConnectException, AdminException
String addJMSPBridgeConnection (int serverId, String urls) throws ConnectException, AdminException
String deleteJMSPBridgeConnection (int serverId, String names) throws ConnectException, AdminException

```

Detailed Description

The Admin class allows to set an administrator connection to a given JORAM server, and provides administration and monitoring methods at a server/platform level.

Constructor Documentation

AdminWrapper.AdminWrapper (Connection cnx) throws JMSEException, ConnectException, AdminException

Creates an administration wrapper for a Joram server. Be careful, if the connection is not started this method will failed with a ConnectException.

Parameters:

cnx	A valid connection to the Joram server.
-----	---

Exceptions:

JMSEException	A problem occurs during initialization.
---------------	---

Member Function Documentation

String AdminWrapper.addAMQPBridgeConnection (int serverId, String urls) throws ConnectException, AdminException

Adds an AMQP server and starts a live connection with it, accessible via the url provided. A server is uniquely identified by the given name. Adding an existing server won't do anything.

Parameters:

serverId	the serverId
urls	the amqp url list identifying the servers separate by space, for example: amqp://user:pass@localhost:5672/?name=serv1 amqp://user:pass@localhost: 5678/?name=serv2

Returns:

the result of the method

Exceptions:

ConnectException	If the connection fails.
AdminException	If the invocation can't be done or fails

final void AdminWrapper.addDomain (String domain, int sid, int port) throws ConnectException, AdminException

Adds a domain to the platform.

The domain will use the default network component "SimpleNetwork".

Parameters:

domain	Name of the added domain.
sid	Id of the router server that gives access to the added domain.
port	Listening port in the added domain of the router server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

final void AdminWrapper.addDomain (String domain, String network, int sid, int port) throws ConnectException, AdminException

Adds a domain to the platform using a specific network component.

Parameters:

domain	Name of the added domain.
network	Classname of the network component to use.

sid	Id of the router server that gives access to the added domain.
port	Listening port in the added domain of the router server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

String AdminWrapper.addJMSBridgeConnection (int serverId, String urls) throws ConnectException, AdminException

Adds a JMS server and starts a live connection with it, accessible via the url provided. A server is uniquely identified by the given name. Adding an existing server won't do anything.

Parameters:

serverId	the serverId
urls	the jms url list identifying the servers separate by space. ex: jndi_url/?name=cnx1&cf=cfName&jndiFactoryClass=com.xxx.yyy&user=user1&pass=pass1&clientID=clientID

Returns:

the result of the method

Exceptions:

ConnectException	If the connection fails.
AdminException	If the invocation can't be done or fails

final void AdminWrapper.addServer (int sid, String host, String domain, int port, String server) throws ConnectException, AdminException

Adds a server to the platform.

The server is configured without any service.

Parameters:

sid	Id of the added server
host	Address of the host where the added server is started
domain	Name of the domain where the server is added
port	Listening port of the server in the specified domain
server	Name of the added server

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

See also:

addServer(int, String, String, int, String, String[], String[])

final void AdminWrapper.addServer (int sid, String host, String domain, int port, String server, String[] services, String[] args) throws ConnectException, AdminException

Adds a server to the platform.

Parameters:

sid	Id of the added server
host	Address of the host where the added server is started
domain	Name of the domain where the server is added
port	Listening port of the server in the specified domain
server	Name of the added server
services	Names of the service to start within the server
args	Services' arguments

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void admin.AdminWrapper.close ()

Closes the underlying requestor.

Queue AdminWrapper.createDeadMQueue (int serverId, String name) throws ConnectException, AdminException

Creates or retrieves a DeadMessageQueue destination on a given JORAM server.

First a destination with the specified name is searched on the given server, if it does not exist it is created.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the queue.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

Deprecated:

No longer needed, any queue can be used as DMQ.

Destination AdminWrapper.createQueue (String name) throws AdminException, ConnectException

Creates or retrieves a queue destination on a given JORAM server.

Parameters:

name	The name of the queue.
------	------------------------

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

See also:

createQueue(int, String, String, Properties)

Destination AdminWrapper.createQueue (int serverId, String name) throws AdminException, ConnectException

Creates or retrieves a queue destination on a given JORAM server.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the queue.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

See also:

createQueue(int, String, String, Properties)

Destination AdminWrapper.createQueue (int serverId, String name, String className, Properties prop) throws ConnectException, AdminException

Creates or retrieves a queue destination on a given JORAM server.

First a destination with the specified name is searched on the given server, if it does not exist it is created.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the queue.
className	The queue class name.
prop	The queue properties.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

Destination AdminWrapper.createTopic (String name) throws AdminException, ConnectException

Creates or retrieves a topic destination on the underlying JORAM server.

Parameters:

name	The name of the topic.
------	------------------------

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

See also:

createTopic(int, String, String, Properties)

Destination AdminWrapper.createTopic (int serverId, String name) throws AdminException, ConnectException

Creates or retrieves a topic destination on a given JORAM server.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the topic.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

See also:

createTopic(int, String, String, Properties)

Destination AdminWrapper.createTopic (int serverId, String name, String className, Properties prop) throws ConnectException, AdminException

Creates or retrieves a topic destination on a given JORAM server.

First a destination with the specified name is searched on the given server, if it does not exist it is created.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the topic.

className	The topic class name.
prop	The topic properties.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

User AdminWrapper.createUser (String name, String password) throws ConnectException, AdminException

Creates or retrieves a user on the underlying JORAM server.

Parameters:

name	Name of the user.
password	Password of the user.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

See also:

createUser(String, String, int, String)

User AdminWrapper.createUser (String name, String password, int serverId) throws ConnectException, AdminException

Creates or retrieves a user on the underlying JORAM server.

Parameters:

name	Name of the user.
password	Password of the user.
serverId	The identifier of the user's server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

See also:

createUser(String, String, int, String)

User AdminWrapper.createUser (String name, String password, String identityClass) throws AdminException, ConnectException

Creates or retrieves a user on the underlying JORAM server.

Parameters:

name	Name of the user.
password	Password of the user.
identityClass	Classname for authentication, by default SimpleIdentity for user/password.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

See also:

createUser(String, String, int, String)

User AdminWrapper.createUser (String name, String password, int serverId, String identityClassName) throws ConnectException, AdminException

Admin method creating a user for a given server and instantiating the corresponding User object. If the user has already been set on this server, the method simply returns the corresponding User object. Its fails if the target server does not belong to the platform, or if a proxy could not be deployed server side for a new user.

Parameters:

name	Name of the user.
password	Password of the user.
serverId	The identifier of the user's server.
identityClassName	By default user/password for SimpleIdentity.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

User AdminWrapper.createUser (String name, String password, int serverId, String identityClassName, Properties prop) throws ConnectException, AdminException

Admin method creating a user for a given server and instantiating the corresponding User object. If the user has already been set on this server, the method simply returns the corresponding User object. Its fails if the target server does not belong to the platform, or if a proxy could not be deployed server side for a new user.

Parameters:

name	Name of the user.
password	Password of the user.
serverId	The identifier of the user's server.
identityClassName	By default user/password for SimpleIdentity.
prop	properties

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

String AdminWrapper.deleteAMQPBridgeConnection (int serverId, String names) throws ConnectException, AdminException

Removes the live connection to the specified AMQP server.

Parameters:

serverId	the serverId
names	the name identifying the server or list of name separate by space

Returns:

the result of the method

Exceptions:

ConnectException	If the connection fails.
AdminException	If the invocation can't be done or fails

String AdminWrapper.deleteJMSPBridgeConnection (int serverId, String names) throws ConnectException, AdminException

Removes the live connection to the specified JMS server.

Parameters:

serverId	the serverId
----------	--------------

names	the name identifying the server or list of name separate by space
-------	---

Returns:

the result of the method

Exceptions:

ConnectException	If the connection fails.
AdminException	If the invocation can't be done or fails

final String AdminWrapper.getConfiguration () throws ConnectException, AdminException

Returns the current servers configuration (a3servers.xml).

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

final Queue AdminWrapper.getDefaultDMQ () throws ConnectException, AdminException

Returns the default dead message queue for the local server, null if not set.

Returns:

The default dead message queue for the local server, null if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

getDefaultDMQ(int)

final Queue AdminWrapper.getDefaultDMQ (int serverId) throws ConnectException, AdminException

Returns the default dead message queue for a given server, null if not set.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Unique identifier of the server.
----------	----------------------------------

Returns:

The default dead message queue for the local server, null if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

final String AdminWrapper.getDefaultDMQId () throws ConnectException, AdminException

Returns the unique identifier of the default dead message queue for the local server, null if not set.

Returns:

The unique identifier of the default dead message queue for the local server, null if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

`getDefaultDMQId(int)`

final String AdminWrapper.getDefaultDMQId (int serverId) throws ConnectException, AdminException

Returns the unique identifier of the default dead message queue for a given server, null if not set. The request fails if the target server does not belong to the platform.

Parameters:

serverId	Unique identifier of the server.
----------	----------------------------------

Returns:

The unique identifier of the default dead message queue for the local server, null if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

final int AdminWrapper.getDefaultThreshold () throws ConnectException, AdminException

Returns the default threshold value for the local server, -1 if not set.

Returns:

The default threshold value for the local server, -1 if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

`getDefaultThreshold(int)`

final int AdminWrapper.getDefaultThreshold (int serverId) throws ConnectException, AdminException

Returns the default threshold value for a given server, -1 if not set. The request fails if the target server does not belong to the platform.

Returns:

The default threshold value for the local server, -1 if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

final Destination [] AdminWrapper.getDestinations () throws ConnectException, AdminException

Returns the list of all destinations that exist on the local server.

Returns:

An array containing all destinations defined on the given server or null if none exists.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	Never thrown.

See also:

[getDestinations\(int\)](#)

final Destination [] AdminWrapper.getDestinations (int serverId) throws ConnectException, AdminException

Returns the list of all destinations that exist on a given server.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Unique identifier of the server.
----------	----------------------------------

Returns:

An array containing all destinations defined on the given server or null if none exists.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

final String [] AdminWrapper.getDomainNames (int serverId) throws ConnectException, AdminException

Returns the list of the domain names that contains the specified server.

Parameters:

serverId	Unique identifier of the server.
----------	----------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

final String AdminWrapper.getLocalHost () throws ConnectException, AdminException

Returns the host name of the server the module is connected to.

Exceptions:

ConnectException	If the admin connection is not established.
AdminException	If the request fails.

See also:

[getLocalServer\(\)](#)

final String AdminWrapper.getLocalName () throws ConnectException, AdminException

Returns the name of the server the module is connected to.

Exceptions:

ConnectException	If the admin connection is not established.
AdminException	If the request fails.

See also:

[getLocalServer\(\)](#)

final Server AdminWrapper.getLocalServer () throws ConnectException, AdminException

Returns the information about the current server: unique identifier, symbolic name and hostname.

Returns:

The description of the server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

final int AdminWrapper.getLocalServerId () throws ConnectException, AdminException

Returns the identifier of the server the module is connected to.

Exceptions:

ConnectException	If the admin connection is not established.
AdminException	If the request fails.

See also:

[getLocalServer\(\)](#)

final Server [] AdminWrapper.getServers () throws ConnectException, AdminException

Returns the list of the platform's servers' identifiers.

Returns:

An array containing the description of all servers.

Exceptions:

ConnectException	
AdminException	

See also:

[getServers\(String\)](#)

final Server [] AdminWrapper.getServers (String domain) throws ConnectException, AdminException

Returns the list of the servers' that belong to the specified domain.

Parameters:

domain	Name of the domain.
--------	---------------------

Returns:

An array containing the description of all servers.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

final int [] AdminWrapper.getServerIds () throws ConnectException, AdminException

Returns the list of the platform's servers' identifiers.

Returns:

An array containing the list of server's identifiers.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

[getServers\(String\)](#)

final int [] AdminWrapper.getServerIds (String domain) throws ConnectException, AdminException

Returns the list of the servers' identifiers that belong to the specified domain

Parameters:

domain	Name of the domain.
--------	---------------------

Returns:

An array containing the list of server's identifiers of the specified domain.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

final String [] AdminWrapper.getServerNames () throws ConnectException, AdminException

Returns the list of the platform's servers' names.

Returns:

An array containing the list of server's names.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

[getServers\(String\)](#)

final String [] AdminWrapper.getServerNames (String domain) throws ConnectException, AdminException

Returns the list of the servers' names that belong to the specified domain

Parameters:

domain	Name of the domain.
--------	---------------------

Returns:

An array containing the list of server's names of the specified domain.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

final Hashtable AdminWrapper.getStatistics () throws ConnectException, AdminException

Returns statistics for the local server.

Returns:

statistics for the local server.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

[getStatistics\(int\)](#)

final Hashtable AdminWrapper.getStatistics (int serverId) throws ConnectException, AdminException

Returns statistics for the the specified server.
The request fails if the target server does not belong to the platform.

Parameters:

serverId	Unique identifier of the server.
----------	----------------------------------

Returns:

the statistics for the the specified server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

final long AdminWrapper.getTimeOutToAbortRequest () throws ConnectException

Returns the maximum time in ms before aborting request.

Returns:

the maximum time in ms before aborting request.

Exceptions:

ConnectException	if the connection is not established.
------------------	---------------------------------------

final User [] AdminWrapper getUsers () throws ConnectException, AdminException

Returns the list of all users that exist on the local server.

Returns:

An array containing all users defined on the local server, or null if none exist.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

getUsers(int)

final User [] AdminWrapper getUsers (int serverId) throws ConnectException, AdminException

Returns the list of all users that exist on a given server.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Unique identifier of the given server.
----------	--

Returns:

An array containing all users defined on the local server, or null if none exist.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

```
String AdminWrapper.invokeStaticServerMethod (int serverId, String className, String methodName, Class<?>[] parameterTypes, Object[] args) throws ConnectException, AdminException
```

Invokes the specified static method with the specified parameters on the chosen server. The parameters types of the invoked method must be java primitive types, the java objects wrapping them or String type.

Parameters:

serverId	the identifier of the server.
className	the name of the class holding the static method
methodName	the name of the invoked method
parameterTypes	the list of parameters
args	the arguments used for the method call

Returns:

the result of the invoked method after applying Object#toString() method

Exceptions:

ConnectException	If the connection fails.
AdminException	If the invocation can't be done or fails

boolean AdminWrapper.isClosed ()

Returns true if the underlying requestor is closed.

Returns:

true if the underlying requestor is closed.

```
final void AdminWrapper.removeDomain (String domain) throws ConnectException, AdminException
```

Removes a domain from the platform.

Parameters:

domain	Name of the domain to remove
--------	------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

final void AdminWrapper.removeServer (int sid) throws ConnectException, AdminException

Removes a server from the platform.

Parameters:

sid	Id of the removed server
-----	--------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

```
final void AdminWrapper.setDefaultDMQ (Queue dmq) throws ConnectException, AdminException
```

Sets a given dead message queue as the default DMQ for the local server (null for unsetting previous DMQ).

Parameters:

dmq	The dmq to be set as the default one.
-----	---------------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

setDefaultDMQ(int, Queue)

final void AdminWrapper.setDefaultDMQ (int serverId, Queue dmq) throws ConnectException, AdminException

Sets a given dead message queue as the default DMQ for a given server (null for unsetting previous DMQ). The request fails if the target server does not belong to the platform.

Parameters:

serverId	The identifier of the server.
dmq	The dmq to be set as the default one.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

final void AdminWrapper.setDefaultDMQId (String dmqlId) throws ConnectException, AdminException

Sets a given dead message queue as the default DMQ for the local server (null for unsetting previous DMQ).

Parameters:

dmqlId	The dmqlId (AgentId) to be set as the default one.
--------	--

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

setDefaultDMQId(int, String)

final void AdminWrapper.setDefaultDMQId (int serverId, String dmqlId) throws ConnectException, AdminException

Sets a given dead message queue as the default DMQ for a given server (null for unsetting previous DMQ). The request fails if the target server does not belong to the platform.

Parameters:

serverId	The identifier of the server.
dmqlId	The dmqlId (AgentId) to be set as the default one.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

final void AdminWrapper.setDefaultThreshold (int threshold) throws ConnectException, AdminException

Sets a given value as the default threshold for the local server (-1 for unsetting previous value).

Parameters:

threshold	The threshold value to be set.
-----------	--------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

setDefaultThreshold(int, int)

final void AdminWrapper.setDefaultThreshold (int serverId, int threshold) throws ConnectException, AdminException

Sets a given value as the default threshold for a given server (-1 for unsetting previous value). The request fails if the target server does not belong to the platform.

Parameters:

serverId	The identifier of the server.
threshold	The threshold value to be set.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

final void AdminWrapper.setTimeOutToAbortRequest (long timeOut) throws ConnectException

Set the maximum time in ms before aborting request.

Parameters:

timeOut	the maximum time in ms before aborting request.
---------	---

Exceptions:

ConnectException	if the connection is not established.
------------------	---------------------------------------

final void AdminWrapper.stopServer () throws ConnectException, AdminException

Stops the platform local server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

See also:

stopServer(int)

final void AdminWrapper.stopServer (int serverId) throws ConnectException, AdminException

Stops a given server of the platform.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Identifier of the server to stop.
----------	-----------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

11.7. ZeroconfJoramServer

Class org.objectweb.joram.client.jms.admin.server.ZeroconfJoramServer

Static Public Member Functions

```
static void main (String[] args) throws Exception
static void stop ()
static void destroy () throws Exception
```

Detailed Description

This class starts a Joram server without almost any configuration. It just needs to know an existing Joram server (host, port) and the root login. The existing server is usually "s0" from the base configuration. These data are specified by 4 environment properties: ADMIN_HOST_NAME, ADMIN_PORT, ROOT_USER_NAME and ROOT_USER_PWD.

This server uses the current directory to store some data. You can specify another directory with the property BASE_DIR_PATH.

The new server is added into the first domain found in the Joram platform. If no domain exists, a first domain D0 is created. Notice that this bootstrap mechanism has been designed for a single domain platform. If you need to build more complex configuration with several domains you must use the raw Joram administration API.

Member Function Documentation

static void ZeroconfJoramServer.destroy () throws Exception [static]

Destroy the created server. Remove it from the global configuration then clean the datas.

Exceptions:

Exception	
-----------	--

static void ZeroconfJoramServer.main (String[] args) throws Exception [static]

Starts a Joram server without any configuration.

static void ZeroconfJoramServer.stop () [static]

Stop the created server.

12. XML Administration Scripts

12.1. Introduction

This feature allows to execute administration operation using an XML script. It is possible to create and bind in JNDI connection factories, destinations and users. The complete DTD is available in [Gitlab](#) and numerous examples are available with the samples.

An XML administration script can be executed through JMX or Web administration console, the JEE JCA connector or the administration API:

```
AdminModule.executeXMLAdmin("joramAdmin.xml");
```

12.2. Administration connection

The execution of an XML script needs an administration connection, currently there is three different elements allowing to establish such a connection:

- LocalAdminModule,
- TcpAdminModule,
- SSLAdminModule.

Additionally you can add property elements to configure the factory's parameters of the underlying connection factory (see example above). Each property is an element with two attributes: name and value.

12.2.1. LocalAdminModule

The LocalAdminModule element establishes a connection with a colocated JMS provider, it needs to define the login and password attributes⁵.

The example below defines an administration connection to a colocated server using the username "root" and the associated password.

```
<LocalAdminModule login="root" password="root"/>
```

12.2.2. TcpAdminModule

The TcpAdminModule element establishes a connection with a remote JMS provider, additionally to login and password it needs to define the hostname (default "localhost") and listen port (default 16010) of the Joram's TCP service.

The example below initializes an administration connection trough TCP to the server located on host "localhost" and listening on port 16010. It fixes the connectingTimer property for this connection to 60 seconds.

```
<TcpAdminModule host="localhost" port="16010" login="root" password="root">
<property name="connectingTimer" value="60"/>
```

⁵ If not defined the default administrator login ("root") and password ("root") are used.

```
</TcpAdminModule>
```

12.2.3. SSLAdminModule

SSLAdminModule establishes a connection with a remote JMS provider using SSL, it needs the same attributes than a TcpAdminModule.

```
<SSLAdminModule host="localhost" port="16010" login="root" password="root">
  <property name="connectingTimer" value="60"/>
</SSLAdminModule>
```

12.3. Naming

If you want to register created objects in a JNDI's repository you have to declare an InitialContext element defining appropriate properties (see example below). Then you can add jndi sub-element specifying the JNDI's name of the object to the declaration.

```
<InitialContext>
  <property name="java.naming.factory.initial"
            value="fr.dyade.aaa.jndi2.client.NamingContextFactory"/>
  <property name="java.naming.factory.host" value="localhost"/>
  <property name="java.naming.factory.port" value="16400"/>
</InitialContext>
```

12.4. ConnectionFactory

The execution of an XML script allows to create and register ConnectionFactory for later use by JMS client. There is three different elements allowing to define a ConnectionFactory:

- LocalConnectionFactory,
- TcpConnectionFactory,
- SSLConnectionFactory.

A ConnectionFactory element can be named (name attribute) for later use in the script (building of clustered destination's for example). It can be completed by:

- property elements to configure the factory's parameters of this connection factory (see section 4.4),
- inInterceptors or outInterceptors to define the list of interceptors (see section 7.1.3).
- a jndi element to specify the JNDI's name of the created ConnectionFactory.

12.4.1. LocalConnectionFactory

A LocalConnectionFactory defines a connection factory to a colocated JMS provider.

The example below defines a ConnectionFactory named "joramCF" allowing a JMS client to connect to a colocated provider, it defines the "cnxPendingTimer" property to 30 seconds, and it adds a message interceptor implemented by class Interceptor to handle each sent message. This administered object is registered in JNDI with name "joramCF".

```
<LocalConnectionFactory name="joramCF">
  <property name="cnxPendingTimer" value="30000"/>
  <outInterceptors>
    <interceptor className="Interceptor"/>
  </outInterceptors>
```

```
</outInterceptors>
<jndi name="joramCF"/>
</LocalConnectionFactory>
```

12.4.2. TcpConnectionFactory

A TcpConnectionFactory defines a connection factory to a remote JMS provider, it needs to define the hostname (default “localhost”) and listen port (default 16010) of the Joram’s TCP service.

The example below defines a ConnectionFactory allowing a JMS client to connect to the provider located on “localhost” and listening on port 16010. This ConnectionFactory is configured with two additional properties “queueMessageReadMax” and “topicAckBufferMax” (see [FactoryParameters](#)). This administered object is registered in JNDI with name “joramCF”.

```
<TcpConnectionFactory name="joramCF" host="localhost" port="16010">
  <property name="queueMessageReadMax" value="10"/>
  <property name="topicAckBufferMax" value="5"/>
  <jndi name="joramCF"/>
</TcpConnectionFactory>
```

12.4.3. SSLConnectionFactory

A SSLConnectionFactory defines a connection factory to a remote JMS provider using SSL, it needs the same attributes than a TcpConnectionFactory.

```
<SSLConnectionFactory name="joramCF" host="localhost" port="16010">
  <jndi name="joramCF"/>
</SSLConnectionFactory>
```

12.5. User

A User element allows the definition of a Joram’s user (see [User](#)), you must at least define name and password properties:

- name: the name of the user needed for later use in the script (handling of destination’s rights for example).
- login, password: login and password for user, if the login is not defined the name is used by default.
- serverId: unique identifier of location server. If not set the user is created on the server the administrator is connected.
- dmq: name of the Dead Message Queue for the user.
- threshold: maximum number of consumption trials for messages in subscription.

The example below defines a simple user named “anonymous” on server #0.

```
<User name="anonymous" login="anonymous" password="anonymous" serverId="0"/>
```

Additional properties can be defined for user through property sub-element. Each property is an element with two attributes: name and value.

Setting a redelivery delay value

When a message is denied you would prefer that to not redeliver it immediately and to handle it later. The redeliveryDelay property allows to define for all user’s subscription the delay in seconds before a redelivery attempt.

Setting interceptors

Through these properties the user can, for example, defines the lists of message’s interceptors for this user. There is two distinct chains of interceptors:

- The first one defined by the property named “jms_joram_interceptors_in” handles each message that’s entering the server (result of a send method on a connection from the selected user).
- The second one defined by the property named “jms_joram_interceptors_out” handles each message that’s exiting the server (result of a receive method on a connection from the selected user).

These two interceptor chains are configurable for each user.

12.6. Destinations

The Queue and Topic elements allow to create and retrieve destinations in the provider. Destination elements can be completed by security settings:

- freeReader: Grants the read right to all users on this destination.
- freeWriter: Grants the write right to all users on this destination.
- reader: Sets a user as a potential reader on the destination, the user name is given in the attribute user.
- writer: Sets a user as a potential writer on the destination, the user name is given in the attribute user.

Additional properties can be defined for destinations, each property is set by a property element with two attributes: name and value. For example, through the “jms_joram_interceptors” properties the user can defines the list of message’s interceptors for this destination. This chain of interceptors handles each message that’s entering the destination, it is configurable for each destination.

The jndi sub-element allows to register the destination in JNDI context, the symbolic name is given in the name attribute.

12.6.1. Queue

A queue definition defines some optional attributes, it can be completed by properties, naming or security elements:

- name: the Joram’s internal name for the queue.
- serverId: unique identifier of location server. If not set the queue is created on the server the administrator is connected.
- dmq: name of the Dead Message Queue for this destination.
- threshold: maximum number of consumption trials for messages in this queue.
- nbMaxMsg: Maximum number of pending messages in the queue.

The optional className attribute allows to define the real class of the destination (see [Specialized destinations](#)). By default a simple queue, org.objectweb.joram.mom.dest.Queue.

```
<Queue name="queue" serverId="0">
    <freeReader/>
    <freeWriter/>
    <jndi name="queue"/>
</Queue>
```

Additional properties can be defined for user through property sub-element. Each property is an element with two attributes: name and value.

Setting a delivery delay value

In some cases it could be useful to not deliver a message immediately⁶. The deliveryDelay property allows to define a minimal delay in milliseconds before the queue tries to deliver this message. If the JMS 2.0 deliveryTime property is fixed on the message, the resulting delay is the maximum of the 2 delays.

Setting a redelivery delay value

When a message is denied you would prefer that to not redeliver it immediately and to handle it later. The redeliveryDelay property allows to define the delay in seconds before a redelivery attempt.

⁶ The JMS 2.0 API allows to set the minimum length of time in milliseconds that must elapse after a message is sent before the JMS provider may deliver the message to a consumer.

12.6.2. Topic

A topic definition defines some optional properties, it can be completed by properties, naming or security elements:

- name: the Joram's internal name for the topic.
- serverId: unique identifier of location server. If not set the topic is created on the server the administrator is connected.
- parent: the internal name of the hierarchical parent of this topic.

The optional className attribute allows to define the real class of the destination (see [Specialized destinations](#)). By default a simple topic, org.objectweb.joram.mom.dest.Topic.

```
<Topic name="topic" serverId="0">
    <freeReader/>
    <freeWriter/>
    <jndi name="topic"/>
</Topic>
```

12.7. JMS bridge destinations

These elements allow to create JMS bridge destinations like acquisition or distribution queue or topic. They are simple factories to simplify the declaration of these complex objects.

The JMS bridge destinations rely on the JMS connection service (see 12.7.6) which purpose is to maintain valid connections with the foreign XMQ JMS provider. This service is normally configured in a3servers.xml file but you can add (resp. remove) dynamically JMS bridge connections (see 12.7.5).

12.7.1. JMSAcquisitionQueue

This element allow the creation of a JMSAcquisitionQueue (see [JMSAcquisitionQueue](#)), additionnaly to the queue's attributes the foreign attribute define the name of the remote JMS destination on which acquire messages. Property elements allow to define the additional properties of this destination.

```
<JMSAcquisitionQueue name="joramInQueue" foreign="foreignQueue" serverId="0">
    <property name="period" value="1000"/>
    <property name="jms.ConnectionUpdatePeriod" value="1000"/>
    <freeReader/>
    <freeWriter/>
    <jndi name="joramInDest"/>
</JMSAcquisitionQueue>
```

12.7.2. JMSDistributionQueue

This element allow the creation of a JMSDistributionQueue (see [JMSDistributionQueue](#)), additionnaly to the queue's attributes the foreign attribute define the name of the remote JMS destination to which messages are forwarded. Property elements allow to define the additional properties of this destination.

```
<JMSDistributionQueue name="joramOutQueue" foreign="foreignQueue" serverId="0">
    <property name="period" value="1000"/>
    <property name="jms.ConnectionUpdatePeriod" value="1000"/>
    <freeWriter/>
```

```
<jndi name="joramOutDest"/>
</JMSDistributionQueue>
```

12.7.3. JMSAcquisitionTopic

This element allow the creation of a JMSAcquisitionTopic (see [JMSAcquisitionTopic](#)), additionnaly to the topic's attributes the foreign attribute define the name of the remote JMS destination on which acquire messages. Property elements allow to define the additional properties of this destination.

```
<JMSAcquisitionTopic name="joramInTopic" foreign="foreignTopic" serverId="0">
  <property name="period" value="1000"/>
  <property name="jms.ConnectionUpdatePeriod" value="1000"/>
  <freeReader/>
  <freeWriter/>
  <jndi name="joramInDest"/>
</JMSAcquisitionTopic>
```

12.7.4. JMSDistributionTopic

This element allow the creation of a JMSDistributionTopic (see [JMSDistributionTopic](#)), additionnaly to the topic's attributes the foreign attribute define the name of the remote JMS destination to which messages are forwarded. Property elements allow to define the additional properties of this destination.

```
<JMSDistributionTopic name="joramOutTopic" foreign="foreignTopic" serverId="0">
  <property name="period" value="1000"/>
  <property name="jms.ConnectionUpdatePeriod" value="1000"/>
  <freeWriter/>
  <jndi name="joramOutDest"/>
</JMSDistributionTopic>
```

12.7.5. JMSBridgeConnection

The JMS bridge destinations rely on a particular Joram service (JMSService) which purpose is to maintain a valid connection with the foreign XMQ JMS providers. Normally this service is declared in the a3servers.xml configuration file as any other Joram's service but JMS bridge connections can be dynamically added using XML administration scripts⁷.

```
<JMSBridgeConnection urls="..." serverId="0"/>
```

A JMSBridgeConnection element needs two attributes:

- serverId: the unique identifier of the related Joram's server.
- urls: the list of URL describing the remote JMS providers.

Each URL is separated by a blank space and describes a ConnectionFactory allowing to connect to the corresponding remote JMS provider. For example the URL below is used in the Joram sample where XMQ is another Joram, it describes:

- The JNDI URL: "scn://localhost:16400/",
- the internal name of the connection: "name=cnx",
- an optional ClientId (arbitrary fixed by the bridge if not set by the client with a risk of conflict),

⁷ The administration API allows to dynamically add /configure / remove these JMS bridge connections.

- the JNDI's name of the ConnectionFactory: "cf=foreignCF",
- the JNDI's factory class: "jndiFactoryClass=...".

```
scn://localhost:16400/?  
name=cnx&clientID=CLIENT1&cf=foreignCF&jndiFactoryClass=fr.dyade.aaa.jndi2.  
client.NamingContextFactory
```

12.7.6. JMSConnectionService

This service must be declared (eventually empty with no JMS bridge connection) in the a3servers.xml file, as any other JORAM service (see a3servers.xml configuration file below).

```
<?xml version="1.0"?>  
<config>  
    <property name="Transaction" value="fr.dyade.aaa.util.NTransaction"/>  
    <server id="0" name="S0" hostname="localhost">  
        <service class="org.objectweb.joram.mom.proxies.ConnectionManager"  
            args="root root"/>  
        <service class="org.objectweb.joram.mom.proxies.tcp.TcpProxyService"  
            args="16010"/>  
        <service class="org.objectweb.joram.mom.dest.jms.JMSConnectionService"  
            args="jndi_url/?name=cnx1&cf=cfName&jndiFactoryClass=com.xxx.yyy&user  
=user1&pass=pwd1&clientID=clientID"/>  
        <service class="fr.dyade.aaa.jndi2.server.JndiServer" args="16400"/>  
    </server>  
</config>
```

The arguments of the JMSConnectionService service are used to configure the way we can access the XMQ servers. The arguments which can be provided are a list of URL separate by a blank space, in order:

- jndi_url** is the JNDI URL which was used to bound the XMQ JMS ConnectionFactory.
- name** represent the connection name, this is used for routing the messages.
- cf** is the name of the connection factory name of XMQ.
- jndiFactoryClass** is the factory of the XMQ jndi.
- user** that should be used by the bridge destination for opening a connection to XMQ. If not provided, the connection will be opened with default identification.
- pass** that should be used by the bridge destination for opening a connection to XMQ. If not provided, the connection will be opened with default password.
- clientID** is provided if XMQ requires the setting of such an identifier on its client connection, this is used for the connection API method setClientID.

Remarque : If the JMS connection service is not declared in the Joram configuration you can add JMS bridge connections but they are not cleanly managed: not stopped at server stop, not restarted at server start.

12.8. Rest / JMS Bridge destinations

These elements allow to create Rest / JMS bridge destinations like acquisition or distribution queue. They are simple factories to simplify the declaration of these complex objects. These destinations require the use of the Rest / JMS API connector on the remote JMS server.

12.8.1. Rest / JMS Acquisition queue

This element allow the creation of a RestAcquisitionQueue (see RestAcquisitionQueue), additionnaly to the queue's attributes the foreign attribute define the name of the remote Rest / JMS destination on which acquire messages (this attribute is required and cannot be null). Property XML elements allow to define the additional properties of this destination (see RestAcquisitionQueue create method).

```
<RestAcquisitionQueue name="queueAcq1" foreign="queue" serverId="1">  
    <property name="rest.host" value="localhost" />  
    <property name="rest.port" value="8989" />
```

```

<property name="rest.timeout" value="5000" />
<property name="rest.idletimeout" value="10000" />
<freeReader />
<jndi name="queueAcq1" />
</RestAcquisitionQueue>

```

12.8.2. Rest / JMS Distribution queue

This element allow the creation of a RDistributionQueue (see [RestDistributionQueue](#)), additionnaly to the queue's attributes the foreign attribute define the name of the remote Rest / JMS destination to which messages are forwarded (this attribute is required and cannot be null). Property XML elements allow to define the additional properties of this destination (see [RestDistributionQueue](#) create method).

```

<RestDistributionQueue name="queueDist1" foreign="queue" serverId="1">
    <property name="rest.host" value="localhost" />
    <property name="rest.port" value="8989" />
    <property name="rest.idletimeout" value="10000" />
    <freeWriter />
    <jndi name="queueDist1" />
</RestDistributionQueue>

```

12.9. AMQP bridge destinations

These elements allow to create AMQP bridge destinations like acquisition or distribution queue or topic. They are simple factories to simplify the declaration of these complex objects.

The AMQP bridge destinations rely on the AMQP connection service (see 12.9.6) which purpose is to maintain valid connections with remote AMQP servers. This service is normally configured in `a3servers.xml` file but you can add (resp. remove) dynamically AMQP bridge connections (see 12.9.5).

12.9.1. AMQPAcquisitionQueue

This element allow the creation of a AMQPAcquisitionQueue (see [AMQPAcquisitionQueue](#)), additionnaly to the queue's attributes the foreign attribute define the name of the remote AMQP destination on which acquire messages. Property elements allow to define the additional properties of this destination.

```

<AMQPAcquisitionQueue name="joramInQueue" foreign="amqpQueue" serverId="0">
    <property name="period" value="1000"/>
    <property name="amqp.Queue.DeclarePassive" value="false"/>
    <property name="amqp.Queue.DeclareExclusive" value="false"/>
    <property name="amqp.Queue.DeclareDurable" value="true"/>
    <property name="amqp.Queue.DeclareAutoDelete" value="false"/>
    <property name="amqp.ConnectionUpdatePeriod" value="1000"/>
    <freeReader/>
    <freeWriter/>

```

```
<jndi name="joramInDest"/>
</AMQPAcquisitionQueue>
```

12.9.2. AMQP Distribution Queue

This element allow the creation of a AMQPDistributionQueue (see [AMQPDistributionQueue](#)), additionnaly to the topic's attributes the foreign attribute define the name of the remote AMQP destination to which messages are forwarded. Property elements allow to define the additional properties of this destination.

```
<AMQPDistributionQueue name="joramOutQueue" foreign="amqpQueue" serverId="0">
  <property name="period" value="1000"/>
  <property name="amqp.Queue.DeclarePassive" value="false"/>
  <property name="amqp.Queue.DeclareExclusive" value="false"/>
  <property name="amqp.Queue.DeclareDurable" value="true"/>
  <property name="amqp.Queue.DeclareAutoDelete" value="false"/>
  <property name="jms.ConnectionUpdatePeriod" value="1000"/>
  <freeWriter/>
  <jndi name="joramOutDest"/>
</AMQPDistributionQueue>
```

12.9.3. AMQPAcquisitionTopic

This element allow the creation of a AMQPAcquisitionTopic (see [AMQPAcquisitionTopic](#)), additionnaly to the queue's attributes the foreign attribute define the name of the remote AMQP destination on which acquire messages. Property elements allow to define the additional properties of this destination.

```
<AMQPAcquisitionTopic name="joramInTopic" foreign="amqpQueue" serverId="0">
  <property name="period" value="1000"/>
  <property name="amqp.Queue.DeclarePassive" value="false"/>
  <property name="amqp.Queue.DeclareExclusive" value="false"/>
  <property name="amqp.Queue.DeclareDurable" value="true"/>
  <property name="amqp.Queue.DeclareAutoDelete" value="false"/>
  <property name="amqp.ConnectionUpdatePeriod" value="1000"/>
  <freeReader/>
  <freeWriter/>
  <jndi name="joramInDest"/>
</AMQPAcquisitionTopic>
```

12.9.4. AMQPDistributionTopic

This element allow the creation of a AMQPDistributionTopic (see [AMQPDistributionTopic](#)), additionnaly to the topic's attributes the foreign attribute define the name of the remote AMQP destination to which messages are forwarded. Property elements allow to define the additional properties of this destination.

```
<AMQPDistributionTopic name="joramOutTopic" foreign="amqpQueue" serverId="0">
    <property name="period" value="1000"/>
    <property name="amqp.Queue.DeclarePassive" value="false"/>
    <property name="amqp.Queue.DeclareExclusive" value="false"/>
    <property name="amqp.Queue.DeclareDurable" value="true"/>
    <property name="amqp.Queue.DeclareAutoDelete" value="false"/>
    <property name="jms.ConnectionUpdatePeriod" value="1000"/>
    <freeWriter/>
    <jndi name="joramOutDest"/>
</AMQPDistributionTopic>
```

12.9.5. AMQPBridgeConnection

The AMQP destinations rely on a particular Joram service which purpose is to maintain a valid connection with the AMQP servers. Normally this service is declared in the a3servers.xml configuration file as any other Joram's service but it can be dynamically configured using XML administration scripts.

```
<AMQPBridgeConnection urls="..." serverId="0"/>
```

An AMQPBridgeConnection element needs two attributes:

- serverId: the unique identifier of the related Joram's server.
- urls: the list of URL describing the remote AMQP servers.

Each URL is separated by a blank space and describes the parameters allowing to connect to the corresponding remote AMQP server: the host and port of the listen socket for the AMQP server. If none is provided, the default host and port are localhost and 5672 (IANA assignment for AMQP).

```
amqp://user:pass@host1:5672/?name=server1 amqp://host2:5672/?name=server2
```

12.9.6. AMQPConnectionService

This service must be declared in the a3servers.xml file, as any other JORAM service (see a3servers.xml configuration file below).

```
<?xml version="1.0"?>
<config>
    <property name="Transaction" value="fr.dyade.aaa.util.NTransaction"/>

    <server id="0" name="S0" hostname="localhost">
        <service class="org.objectweb.joram.mom.proxies.ConnectionManager"
            args="root root"/>
        <service class="org.objectweb.joram.mom.proxies.tcp.TcpProxyService"
            args="16010"/>
        <service class="org.objectweb.joram.mom.dest.amqp.AmqpConnectionService"
            args="amqp://localhost:5672/?name=server1"/>
        <service class="fr.dyade.aaa.jndi2.server.JndiServer" args="16400"/>
    </server>
</config>
```

The arguments of the AmqpConnectionService service are used to configure the host and port of the listen socket for the AMQP server. If none is provided, the default host and port are localhost and 5672 (IANA assignment for AMQP).

12.10. Specialized destinations

These elements allow to create specialized destinations. They are simple factories to simplify the declaration of these complex objects.

12.10.1. CollectorQueue

This element allow the creation of a CollectorQueue (see [CollectorQueue](#)), additionnaly to the queue's attributes the url attribute define the name of the resource to collect. Property elements allow to define the additional properties of this destination.

The CollectorQueue defined in the example below does not regularly collect the resource (acquisition.period=0), the resource acquisition will be triggered explicitly for each received message (the acquisition's parameters can be overloaded in the received message). The produced message is stored in the queue for later consumption.

```
<CollectorQueue name="queue" url="http://www.gnu.org/licenses/lgpl.txt">
  <property name="acquisition.period" value="0"/>

  <freeReader/>
  <freeWriter/>
  <jndi name="queue"/>
</CollectorQueue>
```

12.10.2. CollectorTopic

This element allow the creation of a CollectorTopic (see [CollectorTopic](#)), additionnaly to the queue's attributes the url attribute define the name of the resource to collect. Property elements allow to define the additional properties of this destination.

The CollectorTopic defined in the example below does regularly collect the resource every 5 seconds. The produced messages are immediately forwarded to each suscribers.

```
<CollectorTopic name="topic" url="http://www.gnu.org/licenses/lgpl.txt">
  <property name="acquisition.period" value="5000"/>

  <freeReader/>
  <freeWriter/>
  <jndi name="topic"/>
</CollectorTopic>
```

12.10.3. FtpQueue

12.10.4. MailAcquisitionQueue

12.10.5. MailAcquisitionTopic

12.10.6. MailDistributionQueue

12.10.7. MailDistributionTopic

12.10.8. MonitoringQueue

```
<MonitoringQueue name="MonitoringQueue">
    <freeReader/>
    <freeWriter/>
    <jndi name="MonitoringQueue"/>
</MonitoringQueue>
```

12.10.9. MonitoringTopic

```
<MonitoringTopic name="MonitoringTopic">
    <property name="Joram#0:*" value="DestinationId"/>
    <property name="acquisition.period" value="5000"/>

    <freeReader/>
    <freeWriter/>
    <jndi name="MonitoringTopic"/>
</MonitoringTopic>
```

12.10.10. SchedulerQueue

```
<SchedulerQueue name="queue">
    <freeReader/>
    <freeWriter/>
    <jndi name="queue"/>
</SchedulerQueue>
```

13. OSGi Configuration

Class Reference

13.1. Introduction

Right now, management service with the configuration admin files is based on the OSGi JOnAS “**depmonitor**” service.

The “**depmonitor**” service scans periodically some directories in the aim of deploying OSGi bundles on a JOnAS server. By default, you have to put the application files into the `$JONAS_BASE/deploy` directory in order to deploy them. It is possible to parse another directories by setting the `directories` property in the service configuration (refer to the JOnAS documentation).

The deployment monitor can be configured to detect at runtime if an application is added, removed or changed to respectively deploy it, undeploy it or redeploy it. This functionality can be useful during the development phase.

You can write a single configuration admin file to describe all Joram administration.

From Joram server start to the creation of queues, topics, users and factories. But you can also write several configuration admin files. The deployment service is dynamic, so you can apply a hot administrator configuration. We details thereafter the all possible configurations.

Simple example all in one:

```
<?xml version="1.0" encoding="UTF-8"?>

<configadmin xmlns="http://joram.ow2.org/ns/configadmin/1.0">

    <!-- Start a Joram server, -->
    <!-- the configuration "a3servers.xml" is in jonas.base/conf -->

    <configuration pid="joram.server">
        <property name="sid">0</property>
        <property name="storage">${jonas.base}/work/s0</property>
        <property name="pathToConf">${jonas.base}/conf</property>
        <property name="Transaction">fr.dyade.aaa.util.NTransaction</property>
    </configuration>

    <!-- Create the user anonymous -->
    <factory-configuration pid="joram.user">
        <property name="adminWrapper">ra</property>
    </factory-configuration>

```

```
<property name="name">anonymous</property>
<property name="password">anonymous</property>
</factory-configuration>
<!-- Create the connection factory JCF -->
<factory-configuration pid="joram.connectionfactory">
<property name="type">tcp</property>
<property name="host">localhost</property>
<property name="port">16010</property>
<property name="jndiName">JCF</property>
</factory-configuration>
<!-- Create the connection factory JQCF -->
<factory-configuration pid="joram.connectionfactory">
<property name="type">tcp</property>
<property name="host">localhost</property>
<property name="port">16010</property>
<property name="jndiName">JQCF</property>
</factory-configuration>
<!-- Create the connection factory JTCF -->
<factory-configuration pid="joram.connectionfactory">
<property name="type">tcp</property>
<property name="host">localhost</property>
<property name="port">16010</property>
<property name="jndiName">JTCF</property>
</factory-configuration>
<!-- Create the queue myQueue, and bind it in the Joram jndi -->
<factory-configuration pid="joram.queue">
<property name="adminWrapper">ra</property>
<property name="name">myQueue</property>
<property name="freeReading">true</property>
<property name="freeWriting">true</property>
<property name="jndiName">scn:comp/myQueue</property>
</factory-configuration>
<!-- Create the queue myQueue, and bind it in the JOnAS jndi -->
<factory-configuration pid="joram.queue">
```

```

<property name="adminWrapper">ra</property>
<property name="name">sampleQueue</property>
<property name="freeReading">true</property>
<property name="freeWriting">true</property>
<property name="jndiName">sampleQueue</property>
</factory-configuration>
</configadmin>

```

13.2. JoramManagedService

Class fr.dyade.aaa.agent.services.JoramManagedService
 extends fr.dyade.aaa.agent.services.CommonService
 Implements ManagedService.

Detailed Description

Start a Joram server with a ConfigAdmin file.

Be careful: If you start the Joram resource adapter, the Joram server can be already started and throw an Exception.

The reserved words for properties:

- sid: the server Id (default 0)
- storage: the persistence directory
- pathToConf : the path to the configuration files (a3servers.xml, a3debug.cfg) You can add all needed properties, like Transaction.UseLockFile, ...

A simple example:

```

<configadmin>
  <configuration pid="joram.server">
    <property name="sid">0</property>
    <property name="storage">${jonas.base}/work/s0</property>
    <property name="pathToConf">${jonas.base}/conf</property>
    <property name="Transaction">fr.dyade.aaa.util.NTransaction</property>
    <property name="Transaction.UseLockFile">false</property>
  </configuration>
</configadmin>

```

13.3. Activator

Class org.objectweb.joram.client.osgi.Activator
 Implements BundleActivator.

Detailed Description

The OSGi activator for Destinations, Users and ConnectionFactories ConfigAdmins.

13.4. ServiceConnectionFactory

Class org.objectweb.joram.client.osgi.ServiceConnectionFactory

Implements ManagedServiceFactory.

Detailed Description

Create the connection factories with a ConfigAdmin file.

The reserved words for properties:

- type (required): tcp, local, ssl
- host: default is localhost
- port: default is 16010
- identityClassName: default org.objectweb.joram.shared.security.SimpleIdentity
- inInterceptorClassname: default not set
- outInterceptorClassname: default not set
- jndiName: default not set

A simple example:

```
<configadmin>

<factory-configuration pid="joram.connectionfactory">

    <property name="type">tcp</property>
    <property name="host">localhost</property>
    <property name="port">16010</property>
    <property name="jndiName">cnxFact</property>
</factory-configuration>

</configadmin>
```

13.5. ServiceUser

Class org.objectweb.joram.client.osgi.ServiceUser

Implements ManagedServiceFactory.

Detailed Description

Create a users through a ConfigAdmin file.

The wrapper, need for administration, is an osgi service. This object can be found with the wrapper name (adminWrapper) or the host and port values. If you deploy the Joram resource adapter, the default wrapper name is "ra".

Default values:

- user: anonymous
- password: anonymous
- serverId: 0
- identityClassName: org.objectweb.joram.shared.security.SimpleIdentity

The reserved words for properties:

- adminWrapper: can be used to find the wrapper object need for administration stuff.
- adminHost: can be used to find the wrapper object need for administration stuff.

- adminPort: can be used to find the wrapper object need for administration stuff.
- adminUser: can be used to find the wrapper object need for administration stuff.
- name: the user name.
- password: the user password
- serverId (optionnal): the server id value
- identityClassName: class use for authentication, by default is simple (user, password) can be jaas or an other user implantation.
- SubName: the subscription name
- threshold (optionnal): set the threshold value (default not set), which messages are considered as undeliverable because constantly denied.
- nbMaxMsg (optionnal): set the number max of message store in the subscription (default infinity)
- jndiName (optionnal): rebind this user in Jndi, with the jndiName (default not set)
- dmq (optionnal): Name of the existing dead message queue to set for this user (default not set)
- dmqSid (optionnal): the server id for the dead message queue (default not set)

A simple example:

```
<configadmin>
  <factory-configuration pid="joram.user">
    <property name="adminWrapper">ra</property>
    <property name="name">anonymous</property>
    <property name="password">anonymous</property>
  </factory-configuration>
</configadmin>
```

13.6. ServiceQueue

Class org.objectweb.joram.client.osgi.ServiceQueue
 extends org.objectweb.joram.client.osgi.ServiceDestination
 Implements ManagedServiceFactory.

Detailed Description

Create the queues with a ConfigAdmin file.

The wrapper, need for administration, is an osgi service. This object can be found with the wrapper name (adminWrapper) or the host and port values. If you deploy the Joram resource adapter, the default wrapper name is "ra".

The reserved words for properties:

- adminWrapper: can be used to find the wrapper object need for administration stuff.
- adminHost: can be used to find the wrapper object need for administration stuff.
- adminPort: can be used to find the wrapper object need for administration stuff.
- adminUser: can be used to find the wrapper object need for administration stuff.
- name: the queue name.
- serverId (optionnal): the server id value (default 0)
- freeReading (optionnal): set free reading for all users
- freeWriting (optionnal): set free writing for all users
- readers (optionnal): set the readers users list, users name separate by white space
- writers (optionnal): set the writers users list, users name separate by white space
- threshold (optionnal): set the threshold value (default not set), which messages are considered as undeliverable because constantly denied.
- nbMaxMsg (optionnal): set the number max of message store in this queue (default infinity)
- jndiName (optionnal): rebind this queue in Jndi, with the jndiName (default not set)
- dmq (optionnal): Name of the existing dead message queue to set for this queue (default not set)

- dmqSid (optionnal): the server id for the dead message queue (default not set)

A simple example:

```
<configadmin>

<factory-configuration pid="joram.queue">
    <property name="adminWrapper">ra</property>
    <property name="serverId">0</property>
    <property name="name">queue</property>
    <property name="freeReading">true</property>
    <property name="writers">user1 user2</property>
    <property name="jndiName">queue</property>
</factory-configuration>
</configadmin>
```

13.7. ServiceAcquisitionQueue

Class org.objectweb.joram.client.osgi.ServiceAcquisitionQueue
 extends org.objectweb.joram.client.osgi.ServiceDestination
 Implements ManagedServiceFactory.

Detailed Description

Create a acquisition queue with a ConfigAdmin file.

The wrapper, need for administration, is an osgi service. This object can be found with the wrapper name (adminWrapper) or the host and port values. If you deploy the Joram resource adapter, the default wrapper name is "ra".

The reserved words for properties:

- adminWrapper: can be used to find the wrapper object need for administration stuff.
- adminHost: can be used to find the wrapper object need for administration stuff.
- adminPort: can be used to find the wrapper object need for administration stuff.
- adminUser: can be used to find the wrapper object need for administration stuff.
- name: the queue name.
- serverId (optionnal): the server id value (default 0)
- freeReading (optionnal): set free reading for all users
- freeWriting (optionnal): set free writing for all users
- readers (optionnal): set the readers users list, users name separate by white space
- writers (optionnal): set the writers users list, users name separate by white space
- threshold (optionnal): set the threshold value (default not set), which messages are considered as undeliverable because constantly denied.
- nbMaxMsg (optionnal): set the number max of message store in this queue (default infinity)
- jndiName (optionnal): rebind this queue in Jndi, with the jndiName (default not set)
- dmq (optionnal): Name of the existing dead message queue to set for this queue (default not set)
- dmqSid (optionnal): the server id for the dead message queue (default not set)

A simple example:

```
<configadmin>
    <factory-configuration pid="joram.acquisitionqueue">
```

```

<property name="adminWrapper">ra</property>
<property name="name">acquisitionqueue</property>
<property name="jms.DestinationName">the jndiTargetName</property>
<property name="freeReading">true</property>
<property name="jndiName">acquisitionqueue</property>
</factory-configuration>
</configadmin>

```

13.8. ServiceAliasQueue

Class org.objectweb.joram.client.osgi.ServiceAliasQueue
 extends org.objectweb.joram.client.osgi.ServiceDestination
 Implements ManagedServiceFactory.

Detailed Description

Create an alias queue with a ConfigAdmin file.

The wrapper, need for administration, is an osgi service. This object can be found with the wrapper name (adminWrapper) or the host and port values. If you deploy the Joram resource adapter, the default wrapper name is "ra".

The reserved words for properties:

- adminWrapper: can be used to find the wrapper object need for administration stuff.
- adminHost: can be used to find the wrapper object need for administration stuff.
- adminPort: can be used to find the wrapper object need for administration stuff.
- adminUser: can be used to find the wrapper object need for administration stuff.
- name: the queue name.
- serverId (optionnal): the server id value (default 0)
- freeReading (optionnal): set free reading for all users
- freeWriting (optionnal): set free writing for all users
- readers (optionnal): set the readers users list, users name separate by white space
- writers (optionnal): set the writers users list, users name separate by white space
- threshold (optionnal): set the threshold value (default not set), which messages are considered as undeliverable because constantly denied.
- nbMaxMsg (optionnal): set the number max of message store in this queue (default infinity)
- jndiName (optionnal): rebind this queue in Jndi, with the jndiName (default not set)
- dmq (optionnal): Name of the existing dead message queue to set for this queue (default not set)
- dmqSid (optionnal): the server id for the dead message queue (default not set)

A simple example:

```

<configadmin>
<factory-configuration pid="joram.aliasqueue">
<property name="adminWrapper">ra</property>
<property name="name">aliasqueue</property>
<property name="remoteAgentID">#1.1.1028</property>
<property name="jndiName">aliasqueue</property>
</factory-configuration>

```

```
</configadmin>
```

13.9. ServiceDistributionQueue

Class org.objectweb.joram.client.osgi.ServiceDistributionQueue
 extends org.objectweb.joram.client.osgi.ServiceDestination
 Implements ManagedServiceFactory.

Detailed Description

Create a distribution queue with a ConfigAdmin file.

The wrapper, need for administration, is an osgi service. This object can be found with the wrapper name (adminWrapper) or the host and port values. If you deploy the Joram resource adapter, the default wrapper name is "ra".

The reserved words for properties:

- adminWrapper: can be used to find the wrapper object need for administration stuff.
- adminHost: can be used to find the wrapper object need for administration stuff.
- adminPort: can be used to find the wrapper object need for administration stuff.
- adminUser: can be used to find the wrapper object need for administration stuff.
- name: the queue name.
- serverId (optionnal): the server id value (default 0)
- freeReading (optionnal): set free reading for all users
- freeWriting (optionnal): set free writing for all users
- readers (optionnal): set the readers users list, users name separate by white space
- writers (optionnal): set the writers users list, users name separate by white space
- threshold (optionnal): set the threshold value (default not set), which messages are considered as undeliverable because constantly denied.
- nbMaxMsg (optionnal): set the number max of message store in this queue (default infinity)
- jndiName (optionnal): rebind this queue in Jndi, with the jndiName (default not set)
- dmq (optionnal): Name of the existing dead message queue to set for this queue (default not set)
- dmqSid (optionnal): the server id for the dead message queue (default not set)

A simple example:

```
<configadmin>
<factory-configuration pid="joram.distributionqueue">
  <property name="adminWrapper">ra</property>
  <property name="name">distqueue</property>
  <property name="jms.DestinationName">the jndiTargetName</property>
  <property name="freeWriting">true</property>
  <property name="jndiName">distqueue</property>
</factory-configuration>
</configadmin>
```

13.10. ServiceSchedulerQueue

Class org.objectweb.joram.client.osgi.ServiceSchedulerQueue
 extends org.objectweb.joram.client.osgi.ServiceDestination
 Implements ManagedServiceFactory.

Detailed Description

Create a scheduler queue with a ConfigAdmin file.

The wrapper, need for administration, is an osgi service. This object can be found with the wrapper name (adminWrapper) or the host and port values. If you deploy the Joram resource adapter, the default wrapper name is "ra".

The reserved words for properties:

- adminWrapper: can be used to find the wrapper object need for administration stuff.
- adminHost: can be used to find the wrapper object need for administration stuff.
- adminPort: can be used to find the wrapper object need for administration stuff.
- adminUser: can be used to find the wrapper object need for administration stuff.
- name: the queue name.
- serverId (optionnal): the server id value (default 0)
- freeReading (optionnal): set free reading for all users
- freeWriting (optionnal): set free writing for all users
- readers (optionnal): set the readers users list, users name separate by white space
- writers (optionnal): set the writers users list, users name separate by white space
- threshold (optionnal): set the threshold value (default not set), which messages are considered as undeliverable because constantly denied.
- nbMaxMsg (optionnal): set the number max of message store in this queue (default infinity)
- jndiName (optionnal): rebind this queue in Jndi, with the jndiName (default not set)
- dmq (optionnal): Name of the existing dead message queue to set for this queue (default not set)
- dmqSid (optionnal): the server id for the dead message queue (default not set)

A simple example:

```
<configadmin>
  <factory-configuration pid="joram.schedulerqueue">
    <property name="adminWrapper">ra</property>
    <property name="name">schedulerqueue</property>
    <property name="scheduleDate">10000</property>
    <property name="freeWriting">true</property>
    <property name="jndiName">schedulerqueue</property>
  </factory-configuration>
</configadmin>
```

13.11. ServiceFtpQueue

Class org.objectweb.joram.client.osgi.ServiceFtpQueue
 extends org.objectweb.joram.client.osgi.ServiceDestination
 Implements ManagedServiceFactory.

Detailed Description

Create a ftp queue with a ConfigAdmin file.

The wrapper, need for administration, is an osgi service. This object can be found with the wrapper name (adminWrapper) or the host and port values. If you deploy the Joram resource adapter, the default wrapper name is "ra".

The reserved words for properties:

- adminWrapper: can be used to find the wrapper object need for administration stuff.
- adminHost: can be used to find the wrapper object need for administration stuff.
- adminPort: can be used to find the wrapper object need for administration stuff.
- adminUser: can be used to find the wrapper object need for administration stuff.
- name: the queue name.
- serverId (optionnal): the server id value (default 0)
- freeReading (optionnal): set free reading for all users
- freeWriting (optionnal): set free writing for all users
- readers (optionnal): set the readers users list, users name separate by white space
- writers (optionnal): set the writers users list, users name separate by white space
- threshold (optionnal): set the threshold value (default not set), which messages are considered as undeliverable because constantly denied.
- nbMaxMsg (optionnal): set the number max of message store in this queue (default infinity)
- jndiName (optionnal): rebind this queue in Jndi, with the jndiName (default not set)
- dmq (optionnal): Name of the existing dead message queue to set for this queue (default not set)
- dmqSid (optionnal): the server id for the dead message queue (default not set)

A simple example:

```
<configadmin>
  <factory-configuration pid="joram.ftpqueue">
    <property name="adminWrapper">ra</property>
    <property name="name">ftpqueue</property>
    <property name="user">the user Name</property>
    <property name="pass">the user password</property>
    <property name="path">/home/user</property>
    <property name="freeWriting">true</property>
    <property name="jndiName">ftpqueue</property>
  </factory-configuration>
</configadmin>
```

13.12. ServiceTopic

Class org.objectweb.joram.client.osgi.ServiceTopic
 extends org.objectweb.joram.client.osgi.ServiceDestination
 Implements ManagedServiceFactory.

Detailed Description

Create a topic with a ConfigAdmin file.

The wrapper, need for administration, is an osgi service. This object can be found with the wrapper name (adminWrapper) or the host and port values. If you deploy the Joram resource adapter, the default wrapper name is "ra".

The reserved words for properties:

- adminWrapper: can be used to find the wrapper object need for administration stuff.
- adminHost: can be used to find the wrapper object need for administration stuff.
- adminPort: can be used to find the wrapper object need for administration stuff.
- adminUser: can be used to find the wrapper object need for administration stuff.
- name: the queue name.

- serverId (optionnal): the server id value (default 0)
- freeReading (optionnal): set free reading for all users
- freeWriting (optionnal): set free writing for all users
- readers (optionnal): set the readers users list, users name separate by white space
- writers (optionnal): set the writers users list, users name separate by white space
- threshold (optionnal): set the threshold value (default not set), which messages are considered as undeliverable because constantly denied.
- nbMaxMsg (optionnal): set the number max of message store in this queue (default infinity)
- jndiName (optionnal): rebind this queue in Jndi, with the jndiName (default not set)
- dmq (optionnal): Name of the existing dead message queue to set for this queue (default not set)
- dmqSid (optionnal): the server id for the dead message queue (default not set)

A simple example:

<configadmin>

```
<factory-configuration pid="joram.topic">
  <property name="adminWrapper">ra</property>
  <property name="name">topic</property>
  <property name="freeReading">true</property>
  <property name="freeWriting">true</property>
  <property name="jndiName">topic</property>
</factory-configuration>
</configadmin>
```

13.13. Service Acquisition Topic

Class org.objectweb.joram.client.osgi.ServiceAcquisitionTopic
 extends org.objectweb.joram.client.osgi.ServiceDestination
 Implements ManagedServiceFactory.

Detailed Description

Create a acquisition topic with a ConfigAdmin file.

The wrapper, need for administration, is an osgi service. This object can be found with the wrapper name (adminWrapper) or the host and port values. If you deploy the Joram resource adapter, the default wrapper name is "ra".

The reserved words for properties:

- adminWrapper: can be used to find the wrapper object need for administration stuff.
- adminHost: can be used to find the wrapper object need for administration stuff.
- adminPort: can be used to find the wrapper object need for administration stuff.
- adminUser: can be used to find the wrapper object need for administration stuff.
- name: the queue name.
- serverId (optionnal): the server id value (default 0)
- freeReading (optionnal): set free reading for all users
- freeWriting (optionnal): set free writing for all users
- readers (optionnal): set the readers users list, users name separate by white space
- writers (optionnal): set the writers users list, users name separate by white space
- threshold (optionnal): set the threshold value (default not set), which messages are considered as undeliverable because constantly denied.
- nbMaxMsg (optionnal): set the number max of message store in this queue (default infinity)
- jndiName (optionnal): rebind this queue in Jndi, with the jndiName (default not set)
- dmq (optionnal): Name of the existing dead message queue to set for this queue (default not set)

- dmqSid (optionnal): the server id for the dead message queue (default not set)

A simple example:

```
<configadmin>

<factory-configuration pid="joram.acquisitiontopic">

    <property name="adminWrapper">ra</property>
    <property name="name">acquisitiontopic</property>
    <property name="jms.DestinationName">the jndiTargetName</property>
    <property name="freeReading">true</property>
    <property name="jndiName">acquisitiontopic</property>
</factory-configuration>
</configadmin>
```

13.14. ServiceDistributionTopic

Class org.objectweb.joram.client.osgi.ServiceDistributionTopic
 extends org.objectweb.joram.client.osgi.ServiceDestination
 Implements ManagedServiceFactory.

Detailed Description

Create a distribution topic with a ConfigAdmin file.

The wrapper, need for administration, is an osgi service. This object can be found with the wrapper name (adminWrapper) or the host and port values. If you deploy the Joram resource adapter, the default wrapper name is "ra".

The reserved words for properties:

- adminWrapper: can be used to find the wrapper object need for administration stuff.
- adminHost: can be used to find the wrapper object need for administration stuff.
- adminPort: can be used to find the wrapper object need for administration stuff.
- adminUser: can be used to find the wrapper object need for administration stuff.
- name: the queue name.
- serverId (optionnal): the server id value (default 0)
- freeReading (optionnal): set free reading for all users
- freeWriting (optionnal): set free writing for all users
- readers (optionnal): set the readers users list, users name separate by white space
- writers (optionnal): set the writers users list, users name separate by white space
- threshold (optionnal): set the threshold value (default not set), which messages are considered as undeliverable because constantly denied.
- nbMaxMsg (optionnal): set the number max of message store in this queue (default infinity)
- jndiName (optionnal): rebind this queue in Jndi, with the jndiName (default not set)
- dmq (optionnal): Name of the existing dead message queue to set for this queue (default not set)
- dmqSid (optionnal): the server id for the dead message queue (default not set)

A simple example:

```
<configadmin>

<factory-configuration pid="joram.distributiontopic">

    <property name="adminWrapper">ra</property>
    <property name="name">disttopic</property>
```

```

<property name="jms.DestinationName">the jndiTargetName</property>
<property name="freeWriting">true</property>
<property name="jndiName">disttopic</property>
</factory-configuration>
</configadmin>

```

13.15.ConnectionFactoryMSF

Class org.objectweb.joram.client.osgi.ConnectionFactoryMSF
Implements ManagedServiceFactory.

Detailed Description

Create connection factories with a ConfigAdmin file.

The reserved words for properties:

- className
- host
- port
- reliableClass
- timeout
- identityClassName
- inInterceptorClassname
- outInterceptorClassname
- jndiName

A simple example:

```

<configadmin>
  <factory-configuration
    pid="org.objectweb.joram.client.osgi.ConnectionFactoryMSF">
    <property
      name="className">org.objectweb.joram.client.jms.tcp.TcpConnectionFactory</property>
    <property name="host">localhost</property>
    <property name="port">16010</property>
    <property name="jndiName">JCF</property>
  </factory-configuration>
</configadmin>

```

13.16.DestinationMSF

Class org.objectweb.joram.client.osgi.DestinationMSF
Implements ManagedServiceFactory.

Detailed Description

Create destinations with a ConfigAdmin file.

The reserved words for properties:

- adminWrapper
- adminHost
- adminPort
- adminUser
- name
- className
- serverId
- freeReading
- freeWriting
- readers
- writers
- threshold
- nbMaxMsg
- jndiName
- dmq
- dmqSid

A simple example:

<configadmin>

```
<factory-configuration pid="org.objectweb.joram.client.osgi.DestinationMSF">
    <property name="adminWrapper">ra1</property>
    <property name="serverId">0</property>
    <property name="name">queue</property>
    <property name="className">org.objectweb.joram.mom.dest.Queue</property>
    <property name="freeReading">true</property>
    <property name="writers">user1 user2</property>
    <property name="jndiName">queue</property>
</factory-configuration>
</configadmin>
```

Member Function Documentation

void DestinationMSF.doUpdated (String pid, Dictionary properties)
Create the destination

Parameters:

pid	the pid
properties	the destination properties

13.17. UserMSF

Implements ManagedServiceFactory.

Detailed Description

Create Users with a ConfigAdmin file.

Default values:

- user: anonymous

- password: anonymous
- serverId: 0
- identityClassName: org.objectweb.joram.shared.security.SimpleIdentity

The reserved words for properties:

- adminWrapper
- adminHost
- adminPort
- adminUser
- name
- password
- serverId
- identityClassName
- subName
- threshold
- nbMaxMsg
- jndiName
- dmq
- dmqSid

A simple example:

```
<configadmin>

<factory-configuration pid="org.objectweb.joram.client.osgi.UserMSF">

<property name="adminWrapper">ral</property>
<property name="serverId">0</property>
<property name="name">anonymous</property>
<property name="password">anonymous</property>
</factory-configuration>

</configadmin>
```

14. Shell Commands

Administration commands are now available in Joram to manipulate the server. These commands are packaged in three additional bundles:

- *shell-a3* for commands related to the underlying A3 middleware,
- *shell-mom* for commands related to the MOM,
- and *shell-jndi* for commands related to the embedded implementation of JNDI.

14.1. A3 commands

The *shell-a3* bundle give access to commands allowing to administrate the A3 server on which Joram is running.

- [joram:a3:]close – This command stops the Joram and exits the server application.
- [joram:a3:]engineLoad – This command returns the average number of pending notifications over the last minute, it is an evaluation of the server's load.
- [joram:a3:]garbageRatio – This command gives the garbage ratio when the NG transaction mode is set.
- [joram:a3:]info [options...] – This command shows numerous information about the server, the network (in the case of clustering) and/or the transactional persistence manager. By default this command shows information about all embedded components, the following options are available:
 - -eng – Shows info about the engine,
 - -ngt – Shows info about the transactional persistence manager,
 - -net – Shows info about the network (in the case of clustering).
- [joram:a3:]restartServer – Stops and restarts Joram
- [joram:a3:]startServer – Starts Joram
- [joram:a3:]stopServer – Stops Joram

14.2. MOM commands

The *shell-mom* bundle offers commands allowing to monitor and administrate the destination, subscriptions and users.

- [joram:mom:]clear queue <name>
[joram:mom:]clear subscription <user name> <subscription name>
Clears all pending messages of the queue or subscription.
- [joram:mom:]create queue <name> [options]
[joram:mom:]create topic <name> [options]
Creates a new queue or topic, the following options are available:
 - -sid – Unique identifier of the server on which the destination must be deployed.
 - -ext – Implementation class of the destination.
- [joram:mom:]delete queue <name>
[joram:mom:]delete topic <name>
Deletes the specified destination.
- [joram:mom:]create user <name>
Creates a new user, the command prompts for a password.
- [joram:mom:]delete user <name>
Deletes the specified user.
- [joram:mom:]deleteMsg queue <name> <identifier>
[joram:mom:]deleteMsg subscription <user> <subscription> <identifier>
Delete a pending message from a queue or a subscription.

- [joram:mom:]help [<command name>] – Displays usage information about the specified MOM command. If no command is given, lists all MOM commands.
- [joram:mom:]info queue <name>
[joram:mom:]info topic <name>
[joram:mom:]info subscription <user name> <subscription name>
Gives info about the destination or subscription.
- [joram:mom:]list destination
[joram:mom:]list queue
[joram:mom:]list topic
Lists all destinations defined on the server, or only queues or topics.
- [joram:mom:]list user
Lists all users defined on the server.
- [joram:mom:]list subscription <user name>
Lists subscriptions of the specified user.
- [joram:mom:]lsMsg queue <queue name> [[first]:[last]]
[joram:mom:]lsMsg subscription <user name> <sub. name> [[first]:[last]]
Lists the pending messages of a queue or subscription. Returns messages only messages in the specified interval if it is set, all messages otherwise.
- [joram:mom:]ping – Tells whether this Joram server is running.
- [joram:mom:]queueLoad <queue name> – Gives the average load for the specified queue.
- [joram:mom:]subscriptionLoad <user name> <subscription name> – Gives the average load for the specified subscription.
- [joram:mom:]receiveMsg – Not yet implemented.
- [joram:mom:]sendMsg – Not yet implemented.

14.3. JNDI commands

The *shell-jndi* bundle currently give access only to the `joram:jndi:list` command. This command lists all records in the naming context, giving useful info about the registered destinations and connection factories.

15. JMX API

Class Reference

15.1. JoramAdmin

Class org.objectweb.joram.client.jms.admin.JoramAdmin
Implements AdminIf, JoramAdminMBean

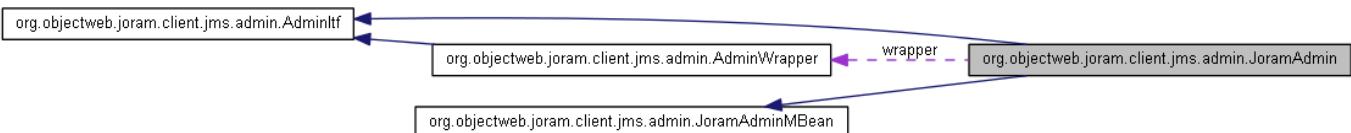


Figure 15.1 - Collaboration diagram for JoramAdmin class

Public Member Functions

```

void exit ()
void close ()
boolean isClosed ()
void setTimeOutToAbortRequest (long timeOut) throws ConnectException
long getTimeOutToAbortRequest () throws ConnectException
void stopServer () throws ConnectException, AdminException
void stopServer (int serverId) throws ConnectException, AdminException
void addServer (int sid, String host, String domain, int port, String server) throws ConnectException,
AdminException
void addServer (int sid, String host, String domain, int port, String server, String[] services, String[] args) throws
ConnectException, AdminException
void removeServer (int sid) throws ConnectException, AdminException
void addDomain (String domain, int sid, int port) throws ConnectException, AdminException
void addDomain (String domain, String network, int sid, int port) throws ConnectException, AdminException
void removeDomain (String domain) throws ConnectException, AdminException
String getConfiguration () throws ConnectException, AdminException
int[] getServersIds () throws ConnectException, AdminException
int[] getServersIds (String domain) throws ConnectException, AdminException
Server[] getServers () throws ConnectException, AdminException
Server[] getServers (String domain) throws ConnectException, AdminException
String[] getDomainNames (int serverId) throws ConnectException, AdminException
String getDefaultDMQId () throws ConnectException, AdminException
String getDefaultDMQId (int serverId) throws ConnectException, AdminException
void setDefaultDMQId (String dmqid) throws ConnectException, AdminException
void setDefaultDMQId (int serverId, String dmqid) throws ConnectException, AdminException
Queue getDefaultDMQ () throws ConnectException, AdminException
Queue getDefaultDMQ (int serverId) throws ConnectException, AdminException
  
```

```

void setDefaultDMQ (Queue dmq) throws ConnectException, AdminException
void setDefaultDMQ (int serverId, Queue dmq) throws ConnectException, AdminException
void setDefaultThreshold (int threshold) throws ConnectException, AdminException
void setDefaultThreshold (int serverId, int threshold) throws ConnectException, AdminException
int getDefaultThreshold () throws ConnectException, AdminException
int getDefaultThreshold (int serverId) throws ConnectException, AdminException
void getLocalDestinations () throws ConnectException, AdminException
Destination[] getDestinations () throws ConnectException, AdminException
void getAllDestinations (int serverId) throws ConnectException, AdminException
Destination[] getDestinations (int serverId) throws ConnectException, AdminException
Destination createQueue (String name) throws AdminException, ConnectException
Destination createQueue (int serverId, String name) throws AdminException, ConnectException
Destination createQueue (int serverId, String name, String className, Properties prop) throws ConnectException,
AdminException
Destination createTopic (String name) throws AdminException, ConnectException
Destination createTopic (int serverId, String name) throws AdminException, ConnectException
Destination createTopic (int serverId, String name, String className, Properties prop) throws ConnectException,
AdminException
User[] getUsers () throws ConnectException, AdminException
User[] getUsers (int serverId) throws ConnectException, AdminException
User createUser (String name, String password) throws AdminException, ConnectException
User createUser (String name, String password, String identityClass) throws AdminException, ConnectException
User createUser (String name, String password, int serverId) throws AdminException, ConnectException
User createUser (String name, String password, int serverId, String identityClass) throws ConnectException,
AdminException
User createUser (String name, String password, int serverId, String identityClassName, Properties prop) throws
ConnectException, AdminException
void executeXMLAdmin (String cfgDir, String cfgFileName) throws Exception
void executeXMLAdmin (String path) throws Exception
void exportRepositoryToFile (String exportDir, String exportFilename) throws AdminException
AdminException
void queueCreate (String name) throws AdminException, ConnectException
void queueCreate (int serverId, String name) throws AdminException, ConnectException
void topicCreate (String name) throws AdminException, ConnectException
void topicCreate (int serverId, String name) throws AdminException, ConnectException
void getLocalUsers () throws ConnectException, AdminException
void getAllUsers (int serverId) throws ConnectException, AdminException
void userCreate (String name, String password) throws AdminException, ConnectException
void userCreate (String name, String password, String identityClass) throws AdminException, ConnectException
void userCreate (String name, String password, int serverId) throws AdminException, ConnectException
void userCreate (String name, String password, int serverId, String identityClass) throws ConnectException,
AdminException
Hashtable getStatistics () throws ConnectException, AdminException
Hashtable getStatistics (int serverId) throws ConnectException, AdminException
String[] getServersNames () throws ConnectException, AdminException
String[] getServersNames (String domain) throws ConnectException, AdminException
Server getLocalServer () throws ConnectException, AdminException
int getLocalServerId () throws ConnectException, AdminException
String getLocalHost () throws ConnectException, AdminException
String getLocalName () throws ConnectException, AdminException
String invokeStaticServerMethod (int serverId, String className, String methodName, Class<?>[] parameterTypes,
Object[] args) throws ConnectException, AdminException
String addAMQPBridgeConnection (int serverId, String urls) throws ConnectException, AdminException
String deleteAMQPBridgeConnection (int serverId, String names) throws ConnectException, AdminException
String addJMSPBridgeConnection (int serverId, String urls) throws ConnectException, AdminException
String deleteJMSPBridgeConnection (int serverId, String names) throws ConnectException, AdminException

```

Static Public Member Functions

`static JoramAdmin doCreate (AbstractConnectionFactory cf, String name, String password, String identityClass)`
`throws ConnectException, AdminException`

Detailed Description

JoramAdmin is the implementation of the interface JoramAdminMBean. It must only be used to allow administration through JMX.

See also:

AdminModule
AdminWrapper

Constructor Documentation

JoramAdmin.JoramAdmin (Connection cnx) throws ConnectException, AdminException, JMSEException

Creates a MBean to administer Joram using the default basename for JMX registering (JoramAdmin(Connection, String)). Be careful, if the connection is not started this method will failed with a ConnectException.

Parameters:

<code>cnx</code>	A valid connection to the Joram server.
------------------	---

Exceptions:

<code>JMSEException</code>	A problem occurs during initialization.
----------------------------	---

See also:

AdminWrapper::AdminWrapper(Connection)

JoramAdmin.JoramAdmin (Connection cnx, String base) throws ConnectException, AdminException, JMSEException

Creates a MBean to administer Joram using the given basename for JMX registering. Be careful, if the connection is not started this method will failed with a ConnectException.

Parameters:

<code>cnx</code>	A valid connection to the Joram server.
<code>base</code>	the basename for registering the MBean.

Exceptions:

<code>JMSEException</code>	A problem occurs during initialization.
----------------------------	---

See also:

AdminWrapper::AdminWrapper(Connection)

Member Function Documentation

String JoramAdmin.addAMQPBridgeConnection (int serverId, String urls) throws ConnectException, AdminException

Adds an AMQP server and starts a live connection with it, accessible via the url provided. A server is uniquely identified by the given name. Adding an existing server won't do anything.

Parameters:

<code>serverId</code>	the serverId
<code>urls</code>	the amqp url list identifying the servers separate by space. ex: amqp://user:pass:5672/?name=serv1 amqp://user:pass:5678/? name=serv2

Returns:

the result of the method

Exceptions:

ConnectException	If the connection fails.
AdminException	If the invocation can't be done or fails

void JoramAdmin.addDomain (String domain, int sid, int port) throws ConnectException, AdminException

Adds a domain to the platform.

The domain will use the default network component "SimpleNetwork".

Parameters:

domain	Name of the added domain.
sid	Id of the router server that gives access to the added domain.
port	Listening port in the added domain of the router server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void JoramAdmin.addDomain (String domain, String network, int sid, int port) throws ConnectException, AdminException

Adds a domain to the platform using a specific network component.

Parameters:

domain	Name of the added domain.
network	Classname of the network component to use.
sid	Id of the router server that gives access to the added domain.
port	Listening port in the added domain of the router server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

String JoramAdmin.addJMSBridgeConnection (int serverId, String urls) throws ConnectException, AdminException

Adds a JMS server and starts a live connection with it, accessible via the url provided. A server is uniquely identified by the given name. Adding an existing server won't do anything.

Parameters:

serverId	the serverId
urls	the jms url list identifying the servers separate by space. ex: jndi_url/?name=cnx1&cf=cfName&jndiFactoryClass=com.xxx.yyy&user=user1&pass=pass1&clientID=clientID

Returns:

the result of the method

Exceptions:

ConnectException	If the connection fails.
AdminException	If the invocation can't be done or fails

void JoramAdmin.addServer (int sid, String host, String domain, int port, String server) throws ConnectException, AdminException

Adds a server to the platform.

The server is configured without any service.

Parameters:

sid	Id of the added server
host	Address of the host where the added server is started
domain	Name of the domain where the server is added
port	Listening port of the server in the specified domain
server	Name of the added server

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void JoramAdmin.addServer (int sid, String host, String domain, int port, String server, String[] services, String[] args) throws ConnectException, AdminException

Adds a server to the platform.

Parameters:

sid	Id of the added server
host	Address of the host where the added server is started
domain	Name of the domain where the server is added
port	Listening port of the server in the specified domain
server	Name of the added server
services	Names of the service to start within the server
args	Services' arguments

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void JoramAdmin.close ()

Closes the underlying requestor.

Destination JoramAdmin.createQueue (String name) throws AdminException, ConnectException

Creates or retrieves a queue destination on the underlying JORAM server, (re)binds the corresponding Queue instance.

Parameters:

name	The name of the queue.
------	------------------------

Returns:

the destination

Exceptions:

AdminException	If the creation fails.
ConnectException	If the connection is closed or broken

See also:

createQueue(int, String)

Destination JoramAdmin.createQueue (int serverId, String name) throws AdminException, ConnectException

Creates or retrieves a queue destination on the underlying JORAM server, (re)binds the corresponding Queue instance.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the queue.

Returns:

the destination

Exceptions:

AdminException	If the creation fails.
ConnectException	if the connection is closed or broken

Destination JoramAdmin.createQueue (int serverId, String name, String className, Properties prop) throws ConnectException, AdminException

Creates or retrieves a queue destination on a given JORAM server.

First a destination with the specified name is searched on the given server, if it does not exist it is created.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Properties:

deliveryDelay	The delivery delay in milliseconds used to wait before delivering a message. If set the resulting delay is the max between this value and the message property.
redeliveryDelay	The re-delivery delay use to wait before re-delivering messages after a deny.
period	The property defining the period of periodic tasks.
jms_joram_interceptors	The property defining the list of interceptors.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the queue.
className	The queue class name.
prop	The queue properties.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

Destination JoramAdmin.createTopic (String name) throws AdminException, ConnectException

Creates or retrieves a topic destination on the underlying JORAM server, (re)binds the corresponding Topic instance.

Parameters:

name	The name of the topic.
------	------------------------

Returns:

the destination

Exceptions:

AdminException	If the creation fails.
ConnectException	if the connection is closed or broken

See also:

`createTopic(int, String)`

Destination JoramAdmin.createTopic (int serverId, String name) throws AdminException, ConnectException

Creates or retrieves a topic destination on the underlying JORAM server, (re)binds the corresponding Topic instance.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the topic.

Returns:

the destination

Exceptions:

AdminException	If the creation fails.
ConnectException	If the connection is closed or broken

Destination JoramAdmin.createTopic (int serverId, String name, String className, Properties prop) throws ConnectException, AdminException

Creates or retrieves a topic destination on a given JORAM server.

First a destination with the specified name is searched on the given server, if it does not exist it is created.

The request fails if the target server does not belong to the platform, or if the destination deployment fails server side.

Properties:

period	The property defining the period of periodic tasks.
jms_joram_interceptors	The property defining the list of interceptors.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the topic.
className	The topic class name.
prop	The topic properties.

Exceptions:

ConnectException	If the admin connection is closed or broken.
AdminException	If the request fails.

User JoramAdmin.createUser (String name, String password) throws AdminException, ConnectException

Creates or retrieves a user on the underlying JORAM server.

Parameters:

name	The login name of the user.
password	The password of the user.

Returns:

the user

Exceptions:

AdminException	If the creation fails.
ConnectException	If the connection fails.

See also:

`createUser(String, String, int, String)`

User `JoramAdmin.createUser (String name, String password, String identityClass) throws AdminException, ConnectException`

Creates or retrieves a user on the underlying JORAM server.

Parameters:

<code>name</code>	The login name of the user.
<code>password</code>	The password of the user.
<code>identityClass</code>	The identity class used for authentication.

Returns:

the user

Exceptions:

<code>AdminException</code>	If the creation fails.
<code>ConnectException</code>	If the connection fails.

See also:

`createUser(String, String, int, String)`

User `JoramAdmin.createUser (String name, String password, int serverId) throws AdminException, ConnectException`

Creates or retrieves a user on the given JORAM server.

Parameters:

<code>name</code>	The login name of the user.
<code>password</code>	The password of the user.
<code>serverId</code>	The unique identifier of the Joram server.

Returns:

the user

Exceptions:

<code>AdminException</code>	If the creation fails.
<code>ConnectException</code>	If the connection fails.

See also:

`createUser(String, String, int, String)`

User `JoramAdmin.createUser (String name, String password, int serverId, String identityClass) throws ConnectException, AdminException`

Creates or retrieves a user on the underlying JORAM server.

Parameters:

<code>name</code>	The login name of the user.
<code>password</code>	The password of the user.
<code>serverId</code>	The unique identifier of the Joram server.
<code>identityClass</code>	The identity class used for authentication.

Returns:

the user

Exceptions:

<code>AdminException</code>	If the creation fails.
<code>ConnectException</code>	If the connection fails.

User JoramAdmin.createUser (String name, String password, int serverId, String identityClassName, Properties prop) throws ConnectException, AdminException

Admin method creating a user for a given server and instantiating the corresponding User object. If the user has already been set on this server, the method simply returns the corresponding User object. It fails if the target server does not belong to the platform, or if a proxy could not be deployed server side for a new user.

Parameters:

name	Name of the user.
password	Password of the user.
serverId	The identifier of the user's server.
identityClassName	By default user/password for SimpleIdentity.
prop	properties

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

String JoramAdmin.deleteAMQPBridgeConnection (int serverId, String names) throws ConnectException, AdminException

Removes the live connection to the specified AMQP server.

Parameters:

serverId	the serverId
names	the name identifying the server or list of name separate by space

Returns:

the result of the method

Exceptions:

ConnectException	If the connection fails.
AdminException	If the invocation can't be done or fails

String JoramAdmin.deleteJMSPBridgeConnection (int serverId, String names) throws ConnectException, AdminException

Removes the live connection to the specified AMQP server.

Parameters:

serverId	the serverId
names	the name identifying the server or list of name separate by space

Returns:

the result of the method

Exceptions:

ConnectException	If the connection fails.
AdminException	If the invocation can't be done or fails

static JoramAdmin JoramAdmin.doCreate (AbstractConnectionFactory cf, String name, String password, String identityClass) throws ConnectException, AdminException [static]

Opens a connection dedicated to administering with the Joram server which parameters are wrapped by a given ConnectionFactory .

Parameters:

cf	The Joram's ConnectionFactory to use for connecting.
name	Administrator's name.

password	Administrator's password.
identityClass	identity class name.

Exceptions:

ConnectException	If connecting fails.
AdminException	If the administrator identification is incorrect.

void JoramAdmin.executeXMLAdmin (String cfgDir, String cfgFileName) throws Exception

This method execute the XML script file that the location is given in parameter.

Be careful, currently this method use the static administration connection through the AdminModule Class.

Parameters:

cfgDir	The directory containing the file.
cfgFileName	The script filename.

void executeXMLAdmin (String path) throws Exception

This method execute the XML script file that the pathname is given in parameter.

Be careful, currently this method use the static administration connection through the AdminModule Class.

Parameters:

path	The script pathname.
------	----------------------

void JoramAdmin.exit ()

Closes the administration connection and unregister the MBean.

void exportRepositoryToFile (String exportDir, String exportFilename) throws AdminException

Export the repository content to an XML file

only the destinations objects are retrieved in this version
xml script format of the admin objects (joramAdmin.xml)

Be careful, currently this method use the static administration connection through the AdminModule Class.

Parameters:

exportDir	target directory where the export file will be put
exportFilename	filename of the export file

Exceptions:

AdminException	if an error occurs
----------------	--------------------

void JoramAdmin.getAllDestinations (int serverId) throws ConnectException, AdminException

This method creates and registers MBeans for all the destinations of the selected server.

The request fails if the target server does not belong to the platform.

Exceptions:

ConnectException	If the connection is closed or broken.
AdminException	Never thrown.

See also:

getDestinations()

void JoramAdmin.getAllUsers (int serverId) throws ConnectException, AdminException

This method creates and registers MBeans for all the users of the selected server.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Unique identifier of the given server.
----------	--

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

String JoramAdmin.getConfiguration () throws ConnectException, AdminException

Returns the current servers configuration (equivalent to the a3servers.xml file).

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

Queue JoramAdmin.getDefaultDMQ () throws ConnectException, AdminException

Returns the default dead message queue for the local server, null if not set.

Returns:

The default dead message queue for the local server, null if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:**getDefaultDMQ(int)**

Queue JoramAdmin.getDefaultDMQ (int serverId) throws ConnectException, AdminException

Returns the default dead message queue for a given server, null if not set.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Unique identifier of the server.
----------	----------------------------------

Returns:

The default dead message queue for the local server, null if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

String JoramAdmin.getDefaultDMQId () throws ConnectException, AdminException

Returns the unique identifier of the default dead message queue for the local server, null if not set.

Returns:

The unique identifier of the default dead message queue for the local server, null if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:**getDefaultDMQId(int)****String JoramAdmin.getDefaultDMQId (int serverId) throws ConnectException, AdminException**

Returns the unique identifier of the default dead message queue for the local server, null if not set.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Unique identifier of the server.
----------	----------------------------------

Returns:

The unique identifier of the default dead message queue for the local server, null if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

int JoramAdmin.getDefaultThreshold () throws ConnectException, AdminException

Returns the default threshold value for the local server, -1 if not set.

Returns:

The default threshold value for the local server, -1 if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

getDefaultThreshold(int)

int JoramAdmin.getDefaultThreshold (int serverId) throws ConnectException, AdminException

Returns the default threshold value for a given server, -1 if not set.

The request fails if the target server does not belong to the platform.

Returns:

The default threshold value for the local server, -1 if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

Destination [] JoramAdmin.getDestinations () throws ConnectException, AdminException

This method creates and registers MBeans for all the destinations on the local server.

Returns:

the destinations tab

Exceptions:

ConnectException	If the connection is closed or broken.
AdminException	Never thrown.

See also:

getDestinations(int)

Destination [] JoramAdmin.getDestinations (int serverId) throws ConnectException, AdminException

This method creates and registers MBeans for all the destinations of the selected server.

The request fails if the target server does not belong to the platform.

Returns:

the destinations tab

Exceptions:

ConnectException	If the connection is closed or broken.
AdminException	Never thrown.

See also:

getDestinations()

String [] JoramAdmin.getDomainNames (int serverId) throws ConnectException, AdminException

Returns the list of the domain names that contains the specified server.

Parameters:

serverId	Unique identifier of the server.
----------	----------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

void JoramAdmin.getLocalDestinations () throws ConnectException, AdminException

This method creates and registers MBeans for all the destinations on the local server.

Exceptions:

ConnectException	If the connection is closed or broken.
AdminException	Never thrown.

See also:

getDestinations(int)

String JoramAdmin.getLocalHost () throws ConnectException, AdminException

Returns the host name of the server the module is connected to.

Exceptions:

ConnectException	If the admin connection is not established.
AdminException	If the request fails.

See also:

getLocalServer()

String JoramAdmin.getLocalName () throws ConnectException, AdminException

Returns the port number of the server the module is connected to.

Exceptions:

ConnectException	If the admin connection is not established.
AdminException	If the request fails.

See also:

getLocalServer()

Server JoramAdmin.getLocalServer () throws ConnectException, AdminException

Returns the information about the current server: unique identifier, symbolic name and hostname.

Returns:

The description of the server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

int JoramAdmin.getLocalServerId () throws ConnectException, AdminException

Returns the identifier of the server the module is connected to.

Exceptions:

ConnectException	If the admin connection is not established.
AdminException	If the request fails.

See also:

getLocalServer()

void JoramAdmin.getLocalUsers () throws ConnectException, AdminException

This method creates and registers MBeans for all the users on the local server.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

getAllUsers(int)

Server [] JoramAdmin.getServers () throws ConnectException, AdminException

Returns the list of the platform's servers' description.

Returns:

An array containing the description of all servers.

Exceptions:

ConnectException	
AdminException	

See also:

getServers(String)

Server [] JoramAdmin.getServers (String domain) throws ConnectException, AdminException

Returns the list of the servers' that belong to the specified domain.

Parameters:

domain	Name of the domain.
--------	---------------------

Returns:

An array containing the description of the corresponding servers.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

int [] JoramAdmin.getServerIds () throws ConnectException, AdminException

Returns the list of the platform's servers' identifiers.

Returns:

An array containing the list of server's identifiers.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

getServersIds(String)

int [] JoramAdmin.getServerIds (String domain) throws ConnectException, AdminException

Returns the list of the servers' identifiers that belong to the specified domain

Parameters:

domain	Name of the domain.
--------	---------------------

Returns:

An array containing the list of server's identifiers of the specified domain.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

String [] JoramAdmin.getServerNames () throws ConnectException, AdminException

Returns the list of the platform's servers' names.

Returns:

An array containing the list of server's names.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

getServers(String)

String [] JoramAdmin.getServerNames (String domain) throws ConnectException, AdminException

Returns the list of the servers' names that belong to the specified domain

Parameters:

domain	Name of the domain.
--------	---------------------

Returns:

An array containing the list of server's names of the specified domain.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

Hashtable JoramAdmin.getStatistics () throws ConnectException, AdminException

Returns statistics for the local server.

Returns:

statistics for the local server.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

getStatistics(int)

Hashtable JoramAdmin.getStatistics (int serverId) throws ConnectException, AdminException

Returns statistics for the the specified server.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Unique identifier of the server.
----------	----------------------------------

Returns:

the statistics for the the specified server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

long JoramAdmin.getTimeOutToAbortRequest () throws ConnectException

Gets the maximum time a command has to complete before it is canceled.

Returns:

the maximum time before a command is canceled

Exceptions:

ConnectException	if the connection is not established.
------------------	---------------------------------------

User [] JoramAdmin getUsers () throws ConnectException, AdminException

This method creates and registers MBeans for all the users on the local server.

Returns:

the users tab

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

getUsers(int)

User [] JoramAdmin getUsers (int serverId) throws ConnectException, AdminException

This method creates and registers MBeans for all the users of the selected server.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Unique identifier of the given server.
----------	--

Returns:

the users tab

Exceptions:

ConnectException	If the connection fails.
------------------	--------------------------

n	
AdminException	If the request fails.

String JoramAdmin.invokeStaticServerMethod (int serverId, String className, String methodName, Class<?>[] parameterTypes, Object[] args) throws ConnectException, AdminException

Invokes the specified static method with the specified parameters on the chosen server. The parameters types of the invoked method must be java primitive types, the java objects wrapping them or String type.

Parameters:

serverId	the identifier of the server.
className	the name of the class holding the static method
methodName	the name of the invoked method
parameterTypes	the list of parameters
args	the arguments used for the method call

Returns:

the result of the invoked method after applying Object#toString() method

Exceptions:

ConnectException	If the connection fails.
AdminException	If the invocation can't be done or fails

boolean JoramAdmin.isClosed ()

Returns true if the underlying requestor is closed.

Returns:

true if the underlying requestor is closed.

void JoramAdmin.queueCreate (String name) throws AdminException, ConnectException

Creates or retrieves a queue destination on the underlying JORAM server, (re)binds the corresponding Queue instance.

Parameters:

name	The name of the queue.
------	------------------------

Exceptions:

AdminException	If the creation fails.
ConnectException	if the connection is closed or broken

See also:

queueCreate(int, String)

void JoramAdmin.queueCreate (int serverId, String name) throws AdminException, ConnectException

Creates or retrieves a queue destination on the underlying JORAM server, (re)binds the corresponding Queue instance.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the queue.

Exceptions:

AdminException	If the creation fails.
ConnectException	if the connection is closed or broken

void JoramAdmin.removeDomain (String domain) throws ConnectException, AdminException

Removes a domain from the platform.

Parameters:

domain	Name of the domain to remove
--------	------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void JoramAdmin.removeServer (int sid) throws ConnectException, AdminException

Removes a server from the platform.

Parameters:

sid	Id of the removed server
-----	--------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void JoramAdmin.setDefaultDMQ (Queue dmq) throws ConnectException, AdminException

Sets a given dead message queue as the default DMQ for the local server (null for unsetting previous DMQ).

Parameters:

dmq	The dmq to be set as the default one.
-----	---------------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

[setDefaultDMQ\(int, Queue\)](#)

void JoramAdmin.setDefaultDMQ (int serverId, Queue dmq) throws ConnectException, AdminException

Sets a given dead message queue as the default DMQ for a given server (null for unsetting previous DMQ). The request fails if the target server does not belong to the platform.

Parameters:

serverId	The identifier of the server.
dmq	The dmq to be set as the default one.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void JoramAdmin.setDefaultDMQId (String dmqid) throws ConnectException, AdminException

Sets a given dead message queue as the default DMQ for the local server (null for unsetting previous DMQ).

Parameters:

dmqid	The dmqid (AgentId) to be set as the default one.
-------	---

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

AdminException	Never thrown.
----------------	---------------

See also:

setDefaultDMQId(int, String)

void JoramAdmin.setDefaultDMQId (int serverId, String dmqlId) throws ConnectException, AdminException

Sets a given dead message queue as the default DMQ for a given server (null for unsetting previous DMQ). The request fails if the target server does not belong to the platform.

Parameters:

serverId	The identifier of the server.
dmqlId	The dmqlId (AgentId) to be set as the default one.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void JoramAdmin.setDefaultThreshold (int threshold) throws ConnectException, AdminException

Sets a given value as the default threshold for the local server (-1 for unsetting previous value).

Parameters:

threshold	The threshold value to be set.
-----------	--------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

setDefaultThreshold(int, int)

void JoramAdmin.setDefaultThreshold (int serverId, int threshold) throws ConnectException, AdminException

Sets a given value as the default threshold for a given server (-1 for unsetting previous value).

The request fails if the target server does not belong to the platform.

Parameters:

serverId	The identifier of the server.
threshold	The threshold value to be set.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void JoramAdmin.setTimeOutToAbortRequest (long timeOut) throws ConnectException

Specifies how much time a command has to complete before If the command does not complete within the specified time, it is canceled and an exception is generated.

Be careful, the value can be changed prior to the connection only using the AdminRequestor.REQUEST_TIMEOUT_PROP property.

Parameters:

timeOut	the maximum time before a command is canceled.
---------	--

Exceptions:

ConnectException	if the connection is not established.
------------------	---------------------------------------

n	
---	--

void JoramAdmin.stopServer () throws ConnectException, AdminException

Stops the platform local server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

See also:

stopServer(int)

void JoramAdmin.stopServer (int serverId) throws ConnectException, AdminException

Stops a given server of the platform.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Identifier of the server to stop.
----------	-----------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void JoramAdmin.topicCreate (String name) throws AdminException, ConnectException

Creates or retrieves a topic destination on the underlying JORAM server, (re)binds the corresponding Topic instance.

Parameters:

name	The name of the topic.
------	------------------------

Exceptions:

AdminException	If the creation fails.
ConnectException	If the connection is closed or broken

See also:

topicCreate(int, String)

void JoramAdmin.topicCreate (int serverId, String name) throws AdminException, ConnectException

Creates or retrieves a topic destination on the underlying JORAM server, (re)binds the corresponding Topic instance.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the topic.

Exceptions:

AdminException	If the creation fails.
ConnectException	If the connection is closed or broken

void JoramAdmin.userCreate (String name, String password) throws AdminException, ConnectException

Creates or retrieves a user on the underlying JORAM server.

Parameters:

name	The login name of the user.
password	The password of the user.

Exceptions:

AdminException	If the creation fails.
ConnectException	If the connection fails.

See also:

userCreate(String, String, int, String)

void JoramAdmin.userCreate (String name, String password, String identityClass) throws AdminException, ConnectException

Creates or retrieves a user on the underlying JORAM server.

Parameters:

name	The login name of the user.
password	The password of the user.
identityClass	The identity class used for authentication.

Exceptions:

AdminException	If the creation fails.
ConnectException	If the connection fails.

See also:

userCreate(String, String, int, String)

void JoramAdmin.userCreate (String name, String password, int serverId) throws AdminException, ConnectException

Creates or retrieves a user on the given JORAM server.

Parameters:

name	The login name of the user.
password	The password of the user.
serverId	The unique identifier of the Joram server.

Exceptions:

AdminException	If the creation fails.
ConnectException	If the connection fails.

See also:

userCreate(String, String, int, String)

void JoramAdmin.userCreate (String name, String password, int serverId, String identityClass) throws ConnectException, AdminException

Creates or retrieves a user on the underlying JORAM server.

Parameters:

name	The login name of the user.
password	The password of the user.
serverId	The unique identifier of the Joram server.
identityClass	The identity class used for authentication.

Exceptions:

AdminException	If the creation fails.
ConnectException	If the connection fails.

n

15.2. JoramAdminConnect

Class org.objectweb.joram.client.jms.admin.JoramAdminConnect

Public Member Functions

```
void connect (String name) throws ConnectException, AdminException
void connect (String name, String host, int port, String user, String pass) throws ConnectException,
AdminException
synchronized void exit (boolean force)
```

Detailed Description

Allows the creation of JoramAdmin instances connected to servers.

Member Function Documentation

void JoramAdminConnect.connect (String name) throws ConnectException, AdminException

Creates an administration connection with default parameters, a JoramAdmin MBean is created and registered in the given domain.

Parameters:

name	The name of the corresponding JMX domain.
------	---

Exceptions:

AdminException	If the creation fails.
ConnectException	if the connection is closed or broken

void JoramAdminConnect.connect (String name, String host, int port, String user, String pass) throws ConnectException, AdminException

Creates an administration connection with given parameters, a JoramAdmin MBean is created and registered.

Parameters:

name	The name of the corresponding JMX domain.
host	The hostname of the server.
port	The listening port of the server.
user	The login identification of the administrator.
pass	The password of the administrator.

Exceptions:

AdminException	If the creation fails.
ConnectException	if the connection is closed or broken

synchronized void JoramAdminConnect.exit (boolean force)

Unregisters the MBean.

Parameters:

force	If true calls System.exit method.
-------	-----------------------------------

15.3. JoramAdminConnectMBean

Interface org.objectweb.joram.client.jms.admin.JoramAdminConnectMBean

Public Member Functions

```
void connect (String name) throws ConnectException, AdminException
void connect (String name, String host, int port, String user, String pass) throws ConnectException,
AdminException
void exit (boolean force)
```

Detailed Description

Allows the creation of JoramAdmin instances connected to servers.

Member Function Documentation

void JoramAdminConnectMBean.connect (String name) throws ConnectException, AdminException

Creates an administration connection with default parameters, a JoramAdmin MBean is created and registered in the given domain.

Parameters:

name	The name of the corresponding JMX domain.
------	---

Exceptions:

AdminException	If the creation fails.
ConnectException	if the connection is closed or broken

void JoramAdminConnectMBean.connect (String name, String host, int port, String user, String pass) throws ConnectException, AdminException

Creates an administration connection with given parameters, a JoramAdmin MBean is created and registered.

Parameters:

name	The name of the corresponding JMX domain.
host	The hostname of the server.
port	The listening port of the server.
user	The login identification of the administrator.
pass	The password of the administrator.

Exceptions:

AdminException	If the creation fails.
ConnectException	if the connection is closed or broken

void JoramAdminConnectMBean.exit (boolean force)

Unregisters the MBean.

Parameters:

force	If true calls System.exit method.
-------	-----------------------------------

15.4. JoramAdminMBean

Interface org.objectweb.joram.client.jms.admin.JoramAdminMBean

Public Member Functions

```

void exit ()
void setTimeOutToAbortRequest (long timeOut) throws ConnectException
long getTimeOutToAbortRequest () throws ConnectException
void stopServer () throws ConnectException, AdminException
void stopServer (int serverId) throws ConnectException, AdminException
void addServer (int sid, String host, String domain, int port, String server) throws ConnectException,
AdminException
void removeServer (int sid) throws ConnectException, AdminException
void addDomain (String domain, int sid, int port) throws ConnectException, AdminException
void addDomain (String domain, String network, int sid, int port) throws ConnectException, AdminException
void removeDomain (String domain) throws ConnectException, AdminException
String getConfiguration () throws ConnectException, AdminException
int[] getServerIds () throws ConnectException, AdminException
int[] getServerIds (String domain) throws ConnectException, AdminException
String[] getDomainNames (int serverId) throws ConnectException, AdminException
String getDefaultDMQId (int serverId) throws ConnectException, AdminException
void setDefaultDMQId (int serverId, String dmqid) throws ConnectException, AdminException
void setDefaultDMQId (String dmqid) throws ConnectException, AdminException
String getDefaultDMQId () throws ConnectException, AdminException
void setDefaultThreshold (int serverId, int threshold) throws ConnectException, AdminException
void setDefaultThreshold (int threshold) throws ConnectException, AdminException
int getDefaultThreshold (int serverId) throws ConnectException, AdminException
int getDefaultThreshold () throws ConnectException, AdminException
void getLocalDestinations () throws ConnectException, AdminException
void getAllDestinations (int serverId) throws ConnectException, AdminException
void queueCreate (String name) throws AdminException, ConnectException
void queueCreate (int serverId, String name) throws AdminException, ConnectException
void topicCreate (String name) throws AdminException, ConnectException
void topicCreate (int serverId, String name) throws AdminException, ConnectException
void getLocalUsers () throws ConnectException, AdminException
void getAllUsers (int serverId) throws ConnectException, AdminException
void userCreate (String name, String password) throws AdminException, ConnectException
void userCreate (String name, String password, String identityClass) throws AdminException, ConnectException
void userCreate (String name, String password, int serverId) throws AdminException, ConnectException
void userCreate (String name, String password, int serverId, String identityClass) throws ConnectException,
AdminException
void exportRepositoryToFile (String exportDir, String exportFilename) throws AdminException
void executeXMLAdmin (String cfgDir, String cfgFileName) throws Exception
void executeXMLAdmin (String path) throws Exception

```

Detailed Description

MBean interface for JoramAdmin.
Member Function Documentation

void JoramAdminMBean.addDomain (String domain, int sid, int port) throws ConnectException, AdminException

Adds a domain to the platform.
The domain will use the default network component "SimpleNetwork".

Parameters:

domain	Name of the added domain.
sid	Id of the router server that gives access to the added domain.
port	Listening port in the added domain of the router server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void JoramAdminMBean.addDomain (String domain, String network, int sid, int port) throws ConnectException, AdminException

Adds a domain to the platform using a specific network component.

Parameters:

domain	Name of the added domain.
network	Classname of the network component to use.
sid	Id of the router server that gives access to the added domain.
port	Listening port in the added domain of the router server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void JoramAdminMBean.addServer (int sid, String host, String domain, int port, String server) throws ConnectException, AdminException

Adds a server to the platform.

The server is configured without any service.

Parameters:

sid	Id of the added server
host	Address of the host where the added server is started
domain	Name of the domain where the server is added
port	Listening port of the server in the specified domain
server	Name of the added server

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void JoramAdminMBean.executeXMLAdmin (String cfgDir, String cfgFileName) throws Exception

This method execute the XML script file that the location is given in parameter.

Be careful, currently this method use the static administration connection through the AdminModule Class.

Parameters:

cfgDir	The directory containing the file.
cfgFileName	The script filename.

void JoramAdminMBean.executeXMLAdmin (String path) throws Exception

This method execute the XML script file that the pathname is given in parameter.

Be careful, currently this method use the static administration connection through the AdminModule Class.

Parameters:

path	The script pathname.
------	----------------------

void JoramAdminMBean.exit ()

Closes the administration connection and unregister the MBean.

void JoramAdminMBean.exportRepositoryToFile (String exportDir, String exportFilename) throws AdminException

Export the repository content to an XML file

- only the destinations objects are retrieved in this version
- xml script format of the admin objects (joramAdmin.xml)

Parameters:

exportDir	target directory where the export file will be put
exportFilename	filename of the export file

Exceptions:

AdminException	if an error occurs
----------------	--------------------

void JoramAdminMBean.getAllDestinations (int serverId) throws ConnectException, AdminException

This method creates and registers MBeans for all the destinations of the selected server.

The request fails if the target server does not belong to the platform.

Exceptions:

ConnectException	If the connection is closed or broken.
AdminException	Never thrown.

See also:

getDestinations()

void JoramAdminMBean.getAllUsers (int serverId) throws ConnectException, AdminException

This method creates and registers MBeans for all the users of the selected server.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Unique identifier of the given server.
----------	--

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

String JoramAdminMBean.getConfiguration () throws ConnectException, AdminException

Returns the current servers configuration (equivalent to the a3servers.xml file).

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

String JoramAdminMBean.getDefaultDmqId (int serverId) throws ConnectException, AdminException

Returns the unique identifier of the default dead message queue for the local server, null if not set.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Unique identifier of the server.
----------	----------------------------------

Returns:

The unique identifier of the default dead message queue for the local server, null if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

String JoramAdminMBean.getDefaultDMQId () throws ConnectException, AdminException

Returns the unique identifier of the default dead message queue for the local server, null if not set.

Returns:

The unique identifier of the default dead message queue for the local server, null if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

getDefaultDMQId(int)

int JoramAdminMBean.getDefaultThreshold (int serverId) throws ConnectException, AdminException

Returns the default threshold value for a given server, -1 if not set.

The request fails if the target server does not belong to the platform.

Returns:

The default threshold value for the local server, -1 if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

int JoramAdminMBean.getDefaultThreshold () throws ConnectException, AdminException

Returns the default threshold value for the local server, -1 if not set.

Returns:

The default threshold value for the local server, -1 if not set.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

getDefaultThreshold(int)

String [] JoramAdminMBean.getDomainNames (int serverId) throws ConnectException, AdminException

Returns the list of the domain names that contains the specified server.

Parameters:

serverId	Unique identifier of the server.
----------	----------------------------------

Exceptions:

ConnectException	If the connection fails.
------------------	--------------------------

AdminException	Never thrown.
----------------	---------------

void JoramAdminMBean.getLocalDestinations () throws ConnectException, AdminException

This method creates and registers MBeans for all the destinations on the local server.

Exceptions:

ConnectException	If the connection is closed or broken.
AdminException	Never thrown.

See also:

getAllDestinations(int)

void JoramAdminMBean.getLocalUsers () throws ConnectException, AdminException

This method creates and registers MBeans for all the users on the local server.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

getAllUsers(int)

int [] JoramAdminMBean.getServerIds () throws ConnectException, AdminException

Returns the list of the platform's servers' identifiers.

Returns:

An array containing the list of server's identifiers.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

getServerIds(String)

int [] JoramAdminMBean.getServerIds (String domain) throws ConnectException, AdminException

Returns the list of the servers' identifiers that belong to the specified domain

Parameters:

domain	Name of the domain.
--------	---------------------

Returns:

An array containing the list of server's identifiers of the specified domain.

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

long JoramAdminMBean.getTimeOutToAbortRequest () throws ConnectException

Gets the maximum time a command has to complete before it is canceled.

Returns:

the maximum time before a command is canceled

Exceptions:

ConnectException	A problem occurs during connection.
------------------	-------------------------------------

void JoramAdminMBean.queueCreate (String name) throws AdminException, ConnectException

Creates or retrieves a queue destination on the underlying JORAM server, (re)binds the corresponding Queue instance.

Parameters:

name	The name of the queue.
------	------------------------

Exceptions:

AdminException	If the creation fails.
ConnectException	if the connection is closed or broken

See also:

queueCreate(int, String)

void JoramAdminMBean.queueCreate (int serverId, String name) throws AdminException, ConnectException

Creates or retrieves a queue destination on the underlying JORAM server, (re)binds the corresponding Queue instance.

Parameters:

serverId	The identifier of the server where deploying the queue.
name	The name of the queue.

Exceptions:

AdminException	If the creation fails.
ConnectException	if the connection is closed or broken

void JoramAdminMBean.removeDomain (String domain) throws ConnectException, AdminException

Removes a domain from the platform.

Parameters:

domain	Name of the domain to remove
--------	------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void JoramAdminMBean.removeServer (int sid) throws ConnectException, AdminException

Removes a server from the platform.

Parameters:

sid	Id of the removed server
-----	--------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

```
void JoramAdminMBean.setDefaultDMQId (int serverId, String dmqlId) throws ConnectException, AdminException
```

Sets a given dead message queue as the default DMQ for a given server (null for unsetting previous DMQ). The request fails if the target server does not belong to the platform.

Parameters:

serverId	The identifier of the server.
dmqlId	The dmqlId (AgentId) to be set as the default one.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

```
void JoramAdminMBean.setDefaultDMQId (String dmqlId) throws ConnectException, AdminException
```

Sets a given dead message queue as the default DMQ for the local server (null for unsetting previous DMQ).

Parameters:

dmqlId	The dmqlId (AgentId) to be set as the default one.
--------	--

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

setDefaultDMQId(int, String)

```
void JoramAdminMBean.setDefaultThreshold (int serverId, int threshold) throws ConnectException, AdminException
```

Sets a given value as the default threshold for a given server (-1 for unsetting previous value). The request fails if the target server does not belong to the platform.

Parameters:

serverId	The identifier of the server.
threshold	The threshold value to be set.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

```
void JoramAdminMBean.setDefaultThreshold (int threshold) throws ConnectException, AdminException
```

Sets a given value as the default threshold for the local server (-1 for unsetting previous value).

Parameters:

threshold	The threshold value to be set.
-----------	--------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	Never thrown.

See also:

setDefaultThreshold(int, int)

void JoramAdminMBean.setTimeOutToAbortRequest (long timeOut) throws ConnectException

Specifies how much time a command has to complete before If the command does not complete within the specified time, it is canceled and an exception is generated.

Parameters:

timeOut	the maximum time before a command is canceled.
---------	--

Exceptions:

ConnectException	A problem occurs during connection.
------------------	-------------------------------------

void JoramAdminMBean.stopServer () throws ConnectException, AdminException

Stops the platform local server.

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

See also:

stopServer(int)

void JoramAdminMBean.stopServer (int serverId) throws ConnectException, AdminException

Stops a given server of the platform.

The request fails if the target server does not belong to the platform.

Parameters:

serverId	Identifier of the server to stop.
----------	-----------------------------------

Exceptions:

ConnectException	If the connection fails.
AdminException	If the request fails.

void JoramAdminMBean.topicCreate (String name) throws AdminException, ConnectException

Creates or retrieves a topic destination on the underlying JORAM server, (re)binds the corresponding Topic instance.

Parameters:

name	The name of the topic.
------	------------------------

Exceptions:

AdminException	If the creation fails.
ConnectException	if the connection is closed or broken

See also:

topicCreate(int, String)

void JoramAdminMBean.topicCreate (int serverId, String name) throws AdminException, ConnectException

Creates or retrieves a topic destination on the underlying JORAM server, (re)binds the corresponding Topic instance.

Parameters:

serverId	The identifier of the server where deploying the topic.
name	The name of the topic.

Exceptions:

AdminException	If the creation fails.
ConnectException	If the connection is closed or broken

void JoramAdminMBean.userCreate (String name, String password) throws AdminException, ConnectException

Creates or retrieves a user on the underlying JORAM server.

Parameters:

name	The login name of the user.
password	The password of the user.

Exceptions:

AdminException	If the creation fails.
ConnectException	If the connection fails.

See also:

[userCreate\(String, String, int, String\)](#)

void JoramAdminMBean.userCreate (String name, String password, String identityClass) throws AdminException, ConnectException

Creates or retrieves a user on the underlying JORAM server.

Parameters:

name	The login name of the user.
password	The password of the user.
identityClass	The identity class used for authentication.

Exceptions:

AdminException	If the creation fails.
ConnectException	If the connection fails.

See also:

[userCreate\(String, String, int, String\)](#)

void JoramAdminMBean.userCreate (String name, String password, int serverId) throws AdminException, ConnectException

Creates or retrieves a user on the given JORAM server.

Parameters:

name	The login name of the user.
password	The password of the user.
serverId	The unique identifier of the Joram server.

Exceptions:

AdminException	If the creation fails.
ConnectException	If the connection fails.

See also:

[userCreate\(String, String, int, String\)](#)

void JoramAdminMBean.userCreate (String name, String password, int serverId, String identityClass) throws ConnectException, AdminException

Creates or retrieves a user on the underlying JORAM server.

Parameters:

name	The login name of the user.
password	The password of the user.
serverId	The unique identifier of the Joram server.
identityClass	The identity class used for authentication.

Exceptions:

AdminException	If the creation fails.
ConnectException	If the connection fails.

15.5. LocalConnections

Class org.objectweb.joram.client.jms.local.LocalConnections

Detailed Description

Class used to check off local connections.

LocalConnectionsMBean

Interface org.objectweb.joram.client.jms.local.LocalConnectionsMBean

Detailed Description

Adds JMX monitoring for local connections.

LocalRequestChannelMBean

Interface org.objectweb.joram.client.jms.local.LocalRequestChannelMBean

Public Member Functions

String getUserName ()

Date getCreationDate ()

void close ()

long getSentCount ()

long getReceivedCount ()

Detailed Description

Adds monitoring for a local connection.

Member Function Documentation**void LocalRequestChannelMBean.close ()**

Closes the connection and unregisters the MBean.

Date LocalRequestChannelMBean.getCreationDate ()

Gets connection creation date.

Returns:

the date of creation of the connection.

long LocalRequestChannelMBean.getReceivedCount ()

Gets the number of replies received on the connection.

Returns:

the number of replies received on the connection.

long LocalRequestChannelMBean.getSentCount ()

Gets the number of requests sent on the connection.

Returns:

the number of requests sent on the connection.

String LocalRequestChannelMBean.getUserName ()

Gets connected user's name.

Returns:

the name of the connected user.

16.Joram-Spring

16.1. Introduction

This feature allows to execute a Joram administration operation using an Spring context configuration XML (applicationContext.xml). It is possible to create, delete, rebind and unbind connection factories, destinations and users. And create and start/stop a collocated Joram server. The complete schema is available on Joram web site. We will describe later in detail all the administrations capabilities of the XML file.

Example the Spring context configuration applicationContext.xml:

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:joram="http://joram.ow2.org/schema/joramns"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jms="http://www.springframework.org/schema/jms"
       xmlns:util="http://www.springframework.org/schema/util"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xsi:schemaLocation="
           http://joram.ow2.org/schema/joramns http://joram.ow2.org/schema/joramns/joram-
           spring-1.0.xsd
           http://www.springframework.org/schema/jms
           http://www.springframework.org/schema/jms/spring-jms-3.1.xsd
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
           http://www.springframework.org/schema/tx
           http://www.springframework.org/schema/tx/spring-tx-3.1.xsd
           http://www.springframework.org/schema/jee
           http://www.springframework.org/schema/jee/spring-jee-3.1.xsd
           http://www.springframework.org/schema/util
           http://www.springframework.org/schema/util/spring-util-3.1.xsd">

    <!-- lets create an embedded Joram server -->
    <joram:joramserver id="server" sid="0" persistent="true"
        pathToConf="/JONAS_BASE/conf" storage="/JONAS_BASE/work/s0" stopServer="false"/>
```

```
<!-- Joram administration to use -->
<joram:admin id="wrapper" user="root" pass="root" host="localhost" port="16010" />

<!-- Joram User to use -->
<joram:user id="user" name="anonymous" password="anonymous" deleteOnStop="true"/>

<!-- Joram queue destination to use -->
<joram:queue id="destination" name="queue1" dmq="DMQ" />

<!-- Joram queue destination to use -->
<joram:queue id="destination2" name="queue2" deleteOnStop="true" />

<!-- Joram tcp connection factory to use -->
<joram:tcpConnectionFactory id="jmsFactory" host="localhost" port="16010" />

<!-- Joram local connection factory to use -->
<joram:localConnectionFactory id="LCF"/>

<!-- The spring jms template for consumer -->
<bean id="consumerJmsTemplate" class="org.springframework.jms.core.JmsTemplate">
    <property name="connectionFactory" ref="CF"/>
    <property name="defaultDestination" ref="destination" />
</bean>

<!-- The spring jms template for producer -->
<bean id="prodJmsTemplate" class="org.springframework.jms.core.JmsTemplate">
    <property name="connectionFactory" ref="LCF"/>
    <property name="defaultDestination" ref="destination" />
</bean>

<!-- a sample POJO which uses a Spring JmsTemplate -->
<bean id="producer" class="org.ow2.joram.spring.sample.Producer" init-method="generateMessages">
    <property name="template">
        <ref bean="prodJmsTemplate" />
    </property>
</bean>
```

```

</property>
<property name="destination">
    <ref bean="destination" />
</property>
<property name="messageCount">
    <value>10</value>
</property>
</bean>

<!-- a sample POJO consumer, this is the Message Driven POJO (MDP) -->
<bean id="consumer" class="org.ow2.joram.spring.sample.Consumer">
    <property name="template" ref="consumerJmsTemplate"/>
</bean>

<!-- JNDI name for connection factory on JMS queues -->
<jee:jndi-lookup id="CF" jndi-name="CF" />

<jms:listener-container connection-factory="CF">
    <jms:listener destination="queue1" ref="consumer" />
</jms:listener-container>
</beans>

```

16.2. The Joram server

16.2.1. JoramServerBeanDefinitionParser

Class org.ow2.joram.spring.JoramServerBeanDefinitionParser
 Implements AbstractSingleBeanDefinitionParser.

Public Member Functions

void createAndStart (short sid, String pathToConf, String storage, boolean persistent)

Protected Member Functions

void doParse (Element element, BeanDefinitionBuilder bean)

Detailed Description

Parser for the joram:joramserver xml entry.

The joram server attributes:

- sid (optionnal): the server id value (default 0)
- persistent (optionnal): true if the Joram server is persistent (default false)
- pathToConf (optionnal): the path to the configuration (a3servers.xml, ...) (default ".")
- storage (optionnal): the path to the persistent directory (default "s+sid")
- stopServer (optionnal): if true the server stop on destroy (default false)

A simple example:

```
<!-- lets create an embedded Joram server -->
<joram:joramserver id="server"
    sid="0"
    persistent="true"
    pathToConf="/JONAS_BASE/conf"
    storage="/JONAS_BASE/work/s0"
    stopServer="false"/>
```

Member Function Documentation

void JoramServerBeanDefinitionParser.doParse (Element element, BeanDefinitionBuilder bean) [protected]

Parse the joram server definition, and create and start the collocated server.

See also:

[org.springframework.beans.factory.xml.AbstractSingleBeanDefinitionParser::doParse\(org.w3c.dom.Element, org.springframework.beans.factory.support.BeanDefinitionBuilder\)](#)

void JoramServerBeanDefinitionParser.createAndStart (short sid, String pathToConf, String storage, boolean persistent)

Create and start a collocated Joram server.

Parameters:

sid	the server id
pathToConf	the path to the configuration (a3servers.xml, ...)
storage	the path to the persistent directory
persistent	true if the Joram server is persistent

16.2.2. JoramServer

Class org.ow2.joram.spring.JoramServer
Implements DisposableBean.

Public Member Functions

void create (short sid, String pathToConf, String storage, boolean collocated, boolean persistent)

void destroy () throws Exception

boolean isStopServer ()

void setStopServer (boolean stopServer)

Detailed Description

Start/Stop a colocated Joram server.

Member Function Documentation

void JoramServer.create (short sid, String pathToConf, String storage, boolean collocated, boolean persistent)

Create and start a Joram server (only if colocated).

Parameters:

sid	the server id
pathToConf	the path to the configuration (a3servers.xml, ...)
storage	the path to the persistent directory
collocated	true if the Joram server is colocated
persistent	true if the Joram server is persistent

void JoramServer.destroy () throws Exception

Stop the Joram server only if the stopServer is true.

See also:

[org.springframework.beans.factory.DisposableBean::destroy\(\)](#)

boolean JoramServer.isStopServer ()

Returns:

the stopServer

void JoramServer.setStopServer (boolean stopServer)

Parameters:

stopServer	the stopServer to set
------------	-----------------------

16.3. The Joram administration wrapper

16.3.1. JoramAdminBeanDefinitionParser

Class org.ow2.joram.spring.JoramAdminBeanDefinitionParser

Implements AbstractSingleBeanDefinitionParser.

Protected Member Functions

Class getBeanClass (Element element)

void doParse (Element element, BeanDefinitionBuilder bean)

Detailed Description

Parser for the “joram:admin” XML entry.

The joram administration attributes:

- user (optionnal): the name of the Joram root (default “root”)
- pass (optionnal): the password of the Joram root (default “root”)
- host (optionnal): the host of the Joram server (default “localhost”)

- port (optionnal): the port of the Joram server (default "16010")

A simple example:

```
<!-- Joram administration to use -->
<joram:admin id="wrapper"
  user="root"
  pass="root"
  host="localhost"
  port="16010" />
```

Member Function Documentation

Class JoramAdminBeanDefinitionParser.getBeanClass (Element element) [protected]

Returns:

the Class of JoramAdmin.

See also:

[org.springframework.beans.factory.xml.AbstractSingleBeanDefinitionParser::getBeanClass\(org.w3c.dom.Element\)](#)

void JoramAdminBeanDefinitionParser.doParse (Element element, BeanDefinitionBuilder bean) [protected]

Parse the joram admin definition, and create the admin wrapper. By default JoramAdmin is collocated. If you set a host / port, the collocated set to false. Add the wrapper property value to the bean.

See also:

[org.springframework.beans.factory.xml.AbstractSingleBeanDefinitionParser::doParse\(org.w3c.dom.Element, org.springframework.beans.factory.support.BeanDefinitionBuilder\)](#)

16.3.2. JoramAdmin

Class org.ow2.joram.spring.JoramAdmin
Implements DisposableBean.

Public Member Functions

void destroy () throws Exception

Static Public Member Functions

static AdminItf getWrapper ()
static void setWrapper (AdminItf wrapper)
static void setConnection (Connection cnx)

Detailed Description

this class holds the Joram wrapper admin and the connection.

Member Function Documentation

static AdminItf JoramAdmin.getWrapper () [static]

Returns:

the admin wrapper

static void JoramAdmin.setWrapper (AdminItf wrapper) [static]

Parameters:

wrapper	the admin wrapper to set
---------	--------------------------

static void JoramAdmin.setConnection (Connection cnx) [static]

Parameters:

cnx	the admin connection to set
-----	-----------------------------

void JoramAdmin.destroy () throws Exception

Close the admin wrapper and the connection.

See also:

[org.springframework.beans.factory.DisposableBean::destroy\(\)](#)

16.4. The Joram connection factory

16.4.1. JoramLocalConnectionFactoryBeanDefinitionParser

Class org.ow2.joram.spring.JoramLocalConnectionFactoryBeanDefinitionParser
Implements AbstractSingleBeanDefinitionParser.

Protected Member Functions

void doParse (Element element, BeanDefinitionBuilder bean)

Detailed Description

Parser to the joram:localConnectionFactory xml entry.

The joram local connection factory attributes:

- no attribute for this entry.

A simple example:

<!-- Joram local connection factory to use -->
<joram:localConnectionFactory id="LCF"/>

Member Function Documentation

void JoramLocalConnectionFactoryBeanDefinitionParser.doParse (Element element, BeanDefinitionBuilder bean) [protected]

Create a local connection factory. Add the connectionFactory property value to the bean.

See also:

[org.springframework.beans.factory.xml.AbstractSingleBeanDefinitionParser::doParse\(org.w3c.dom.Element, org.springframework.beans.factory.support.BeanDefinitionBuilder\)](#)

16.4.2. JoramTcpConnectionFactoryBeanDefinitionParser

Class [org.ow2.joram.spring.JoramTcpConnectionFactoryBeanDefinitionParser](#)
Implements [AbstractSingleBeanDefinitionParser](#).

Protected Member Functions

void doParse (Element element, BeanDefinitionBuilder bean)

Detailed Description

Parser to the joram:tcpConnectionFactory xml entry.

The joram TCP connection factory attributes:

- host (optionnal): the host of the Joram server (default “localhost”)
- port (optionnal): the port of the Joram server (default “16010”)

A simple example:

```
<!-- Joram tcp connection factory to use -->
<joram:tcpConnectionFactory id="jmsFactory"
    host="localhost"
    port="16010" />
```

Member Function Documentation

void JoramTcpConnectionFactoryBeanDefinitionParser.doParse (Element element, BeanDefinitionBuilder bean) [protected]

Create a TCP connection factory. Add this connectionFactory property to the bean

See also:

[org.springframework.beans.factory.xml.AbstractSingleBeanDefinitionParser::doParse\(org.w3c.dom.Element, org.springframework.beans.factory.support.BeanDefinitionBuilder\)](#)

16.4.3. JoramConnectionFactory

Class [org.ow2.joram.spring.JoramConnectionFactory](#)
Implements [FactoryBean](#), [DisposableBean](#).

Public Member Functions

void setConnectionFactory (ConnectionFactory connectionFactory)

Detailed Description

This class holds the connection factory (local or tcp).

Member Function Documentation

void JoramConnectionFactory.setConnectionFactory (ConnectionFactory connectionFactory)

Parameters:

connectionFactor	the connection factory to set
y	

16.5. The Joram destinations

16.5.1. JoramQueueBeanDefinitionParser

Class org.ow2.joram.spring.JoramQueueBeanDefinitionParser
Implements AbstractSingleBeanDefinitionParser.

Protected Member Functions

void doParse (Element element, BeanDefinitionBuilder bean)

Detailed Description

Parser to the joram:queue xml entry.

The joram queue attributes:

- name (required): the queue name.
- sid (optionnal): the server id where create the queue (default “0”)
- dmq (optionnal): the dead message queue name
- dmqSid (optionnal): the dmq server id (default “sid”)
- deleteOnStop (optionnal): if true the queue can be delete on destroy (default “false”)

A simple example:

```
<!-- Joram queue destination to use -->
<joram:queue id="destination"
  name="myQueue"
  dmq="DMQ" />
```

Member Function Documentation

void JoramQueueBeanDefinitionParser.doParse (Element element, BeanDefinitionBuilder bean) [protected]

Create a Joram queue. Add the destination property value to the bean.

See also:

org.springframework.beans.factory.xml.AbstractSingleBeanDefinitionParser::doParse(org.w3c.dom.Element, org.springframework.beans.factory.support.BeanDefinitionBuilder)

16.5.2. *JoramTopicBeanDefinitionParser*

Class org.ow2.joram.spring.JoramTopicBeanDefinitionParser
Implements AbstractSingleBeanDefinitionParser.

Protected Member Functions

void doParse (Element element, BeanDefinitionBuilder bean)

Detailed Description

Parser to the joram:topic xml entry.

The joram topic attributes:

- name (required): the topic name.
- sid (optionnal): the server id where create the topic (default “0”)
- dmq (optionnal): the dead message queue name
- dmqSid (optionnal): the dmq server id (default “sid”)
- deleteOnStop (optionnal): if true the topic can be delete on destroy (default “false”)

A simple example:

```
<!-- Joram topic destination to use -->
<joram:topic id="destination"
  name="myTopic"
  dmq="DMQ" />
```

Member Function Documentation

void JoramTopicBeanDefinitionParser.doParse (Element element, BeanDefinitionBuilder bean) [protected]

Create a Joram topic. Add the destination property value to the bean.

See also:

org.springframework.beans.factory.xml.AbstractSingleBeanDefinitionParser::doParse(org.w3c.dom.Element, org.springframework.beans.factory.support.BeanDefinitionBuilder)

16.5.3. *JoramDestination*

Class org.ow2.joram.spring.JoramDestination
Implements FactoryBean, DisposableBean.

Public Member Functions

void setDestination (Destination dest)
boolean isDeleteOnStop ()
void setDeleteOnStop (boolean deleteOnStop)

Detailed Description

This class holds the Joram destination. If set the destination can be deleted on destroy.

Member Function Documentation

void JoramDestination.setDestination (Destination dest)

Parameters:

dest	the Joram destination to set
------	------------------------------

boolean JoramDestination.isDeleteOnStop ()

Returns:

the deleteOnStop

void JoramDestination.setDeleteOnStop (boolean deleteOnStop)

Parameters:

deleteOnStop	the deleteOnStop to set
--------------	-------------------------

16.6. The Joram users

16.6.1. JoramUserBeanDefinitionParser

Class org.ow2.joram.spring.JoramUserBeanDefinitionParser
Implements AbstractSingleBeanDefinitionParser.

Protected Member Functions

void doParse (Element element, BeanDefinitionBuilder bean)

Detailed Description

Parser to the joram:user xml entry.

The joram user attributes:

- sid (optionnal): the server id where create the user (default “0”)
- name (optionnal): the user name (default “anonymous”)
- password (optionnal): the user password (default “anonymous”)
- deleteOnStop (optionnal): if true the user can be delete on destroy (default “false”)

A simple example:

```
<!-- Joram User to use -->
<joram:user id="user"
    name="myUser"
    password="myUser"
    deleteOnStop="true"/>
```

Member Function Documentation

void JoramUserBeanDefinitionParser.doParse (Element element, BeanDefinitionBuilder bean) [protected]

Create a Joram user. Add the user property value to the bean.

See also:

[org.springframework.beans.factory.xml.AbstractSingleBeanDefinitionParser::doParse\(org.w3c.dom.Element, org.springframework.beans.factory.support.BeanDefinitionBuilder\)](#)

16.6.2. JoramUser

Class org.ow2.joram.spring.JoramUser
Implements FactoryBean, DisposableBean.

Public Member Functions

void setUser (User user)
 Object getObject () throws Exception
 void destroy () throws Exception
 boolean isDeleteOnStop ()
 void setDeleteOnStop (boolean deleteOnStop)

Detailed Description

This class holds the Joram user. If set the user can be deleted on destroy.

Member Function Documentation

void JoramUser.setUser (User user)

Parameters:

user	the user to set
------	-----------------

Object JoramUser.getObject () throws Exception

Returns:

the user

See also:

[org.springframework.beans.factory.FactoryBean::getObject\(\)](#)

void JoramUser.destroy () throws Exception

delete the user if the deleteOnStop is set.

See also:

[org.springframework.beans.factory.DisposableBean::destroy\(\)](#)

boolean JoramUser.isDeleteOnStop ()

Returns:

the deleteOnStop

void JoramUser.setDeleteOnStop (boolean deleteOnStop)

Parameters:

deleteOnStop	the deleteOnStop to set
--------------	-------------------------

16.7. Tests configuration

16.7.1. JoramTestServerBeanDefinitionParser

Class org.ow2.joram.spring.JoramTestServerBeanDefinitionParser
Implements AbstractSingleBeanDefinitionParser.

Public Member Functions

void createAndStart (short sid, String port) throws Exception

Detailed Description

Parser to the joram:testserver xml entry.

This if all in one entry, non configuration files, ...

- Create and start a colocated Joram server on localhost.
- Create the Joram administration wrapper
- Create the anonymous user

The joram user attributes:

- sid (optionnal): the server id (default "0")
- port (optionnal): the server port (default "16010")

A simple example:

```
<!-- lets create an embedded Joram server for test, create a user "anonymous" and
create a colocated admin wrapper -->
<joram:testserver id="server" />
```

Member Function Documentation

void JoramTestServerBeanDefinitionParser.createAndStart (short sid, String port) throws Exception

Parameters:

sid	The server id
port	The server port

Exceptions:

Exception

16.8. Running Joram-spring example

This sample illustrates the use of Joram's description in Spring xml descriptors (applicationContext.xml). We use the jetty server to deploy the generated war.

This sample configuration is made of one joram server, this server hosting a queue and a user. The platform is run in non persistent mode on the “localhost” host. A producer send 10 messages and a listener, a Spring message driven POJO (MDP), receive the messages.

This sample code is located in the joram gitlab:

- <https://gitlab.ow2.org/joram/joram/tree/master/examples/joram/spring/sample>

In order to run the demo described below you must extract the code example from gitlab and go to the examples/joram/spring/sample directory.

16.8.1. The webapp descriptors (web.xml)

Set the spring context config location.

```
<!-- Spring param -->

<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring-config/applicationContext.xml</param-value>
</context-param>
```

Bootstrap listener to start up and shut down Spring's root [WebApplicationContext](#). Simply delegates to ContextLoader as well as to [ContextCleanupListener](#).

```
<!-- Spring start context Listener -->

<listener>
    <display-name>spring context loader</display-name>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

16.8.2. The Spring context configuration descriptors

File “applicationContext.xml”, set the xsd schema location.

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:joram="http://joram.ow2.org/schema/joramns"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jms="http://www.springframework.org/schema/jms"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
           http://joram.ow2.org/schema/joramns http://joram.ow2.org/schema/joramns/joram-spring-1.0.xsd
           http://www.springframework.org/schema/jms
           http://www.springframework.org/schema/jms/spring-jms-3.1.xsd
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.1.xsd"
```

```
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-3.1.xsd">
```

For this example we use the Joram test server element, run a colocated joram server and create a anonymous user.

```
<joram:testserver id="server" />
```

This Joram element create the joram “queue1” queue.

```
<joram:queue id="destination" name="queue1" />
```

This Joram element create a Joram local connection factory.

```
<joram:localConnectionFactory id="LCF"/>
```

This Spring element create a JMS template for the consumer. The `JmsTemplate` class is the central class in the Spring JMS core package. It simplifies the use of JMS since it handles the creation and release of resources when sending or synchronously receiving messages.

```
<bean id="consumerJmsTemplate" class="org.springframework.jms.core.JmsTemplate">
    <property name="connectionFactory" ref="LCF"/>
    <property name="defaultDestination" ref="destination" />
</bean>
```

This Spring element create a JMS template for the producer.

```
<bean id="prodJmsTemplate" class="org.springframework.jms.core.JmsTemplate">
    <property name="connectionFactory" ref="LCF"/>
    <property name="defaultDestination" ref="destination" />
</bean>
```

This Spring element create the producer and set the destination, jms template and the number of messages to send. It is a POJO see the code “org.ow2.joram.spring.sample.Producer”.

```
<bean id="producer" class="org.ow2.joram.spring.sample.Producer" init-method="generateMessages">
    <property name="template">
        <ref bean="prodJmsTemplate" />
    </property>
    <property name="destination">
        <ref bean="destination" />
    </property>
    <property name="messageCount">
        <value>10</value>
    </property>
</bean>
```

This Spring element create the consumer. It is a POJO that needs the Jms template. This POJO implement the `javax.jms.MessageLitener` interface.

```
<bean id="consumer" class="org.ow2.joram.spring.sample.Consumer">
    <property name="template" ref="consumerJmsTemplate"/>
```

```
</bean>
```

For asynchronous reception similar to Java EE's message-driven bean style, Spring provides a number of message listener containers that are used to create Message-Driven POJOs (MDPs).

```
<jms:listener-container cache="none" connection-factory="LCF" concurrency="5">
    <jms:listener destination="queue1" ref="consumer" />
</jms:listener-container>
```

16.8.3. Running the example

- mvn jetty:run-war

```
Producer$1.createMessage : Sending message: Message number is 0.
Producer$1.createMessage : Sending message: Message number is 1.
Producer$1.createMessage : Sending message: Message number is 2.
Producer$1.createMessage : Sending message: Message number is 3.
Producer$1.createMessage : Sending message: Message number is 4.
Producer$1.createMessage : Sending message: Message number is 5.
Producer$1.createMessage : Sending message: Message number is 6.
Producer$1.createMessage : Sending message: Message number is 7.
Producer$1.createMessage : Sending message: Message number is 8.
Producer$1.createMessage : Sending message: Message number is 9.
Consumer.onMessage : Processed message Message number is 0.
Consumer.onMessage : Processed message Message number is 1.
Consumer.onMessage : Processed message Message number is 2.
Consumer.onMessage : Processed message Message number is 3.
Consumer.onMessage : Processed message Message number is 4.
Consumer.onMessage : Processed message Message number is 5.
Consumer.onMessage : Processed message Message number is 6.
Consumer.onMessage : Processed message Message number is 7.
Consumer.onMessage : Processed message Message number is 8.
Consumer.onMessage : Processed message Message number is 9.
```

17. Exceptions

17.1. JMS Exceptions

All these exceptions inherits from JMSEException class.

JMSEException	This is the root class of all JMS API exceptions.
IllegalStateException	This exception is thrown when a method is invoked at an illegal or inappropriate time or if the provider is not in an appropriate state for the requested operation.
InvalidClientIDException	This exception must be thrown when a client attempts to set a connection's client ID to a value that is rejected by a provider.
InvalidDestinationException	This exception must be thrown when a destination either is not understood by a provider or is no longer valid.
InvalidSelectorException	This exception must be thrown when a JMS client attempts to give a provider a message selector with invalid syntax.
JMSecurityException	This exception must be thrown when a provider rejects a user name/password submitted by a client.
MessageEOFException	This exception must be thrown when an unexpected end of stream has been reached when a StreamMessage or BytesMessage is being read.
MessageFormatException	This exception must be thrown when a JMS client attempts to use a data type not supported by a message or attempts to read data in a message as the wrong type.
MessageNotReadableException	This exception must be thrown when a JMS client attempts to read a write-only message.
MessageNotWriteableException	This exception must be thrown when a JMS client attempts to write to a read-only message.
ResourceAllocationException	This exception is thrown when a provider is unable to allocate the resources required by a method.
TransactionInProgressException	This exception is thrown when an operation is invalid because a transaction is in progress.
TransactionRolledBackException	This exception must be thrown when a call to Session.commit results in a rollback of the current transaction.

Tableau 17.1 - JMS exception's list

17.2. Administration Exceptions

All these exceptions inherits from AdminException class, they are defined in the org.objectweb.joram.client.jms.admin package.

AdminException	This is the root class of all Joram administration API exceptions.
NameAlreadyUsedException	

ServerIdAlreadyUsedException	
StartFailureException	
UnknownServerException	

Tableau 17.2 Joram administration exception's list

18. JMX indicators

18.1. AgentServer

18.1.1. AgentServer

ObjectName: "AgentServer:server=AgentServer#0"⁸

Attributes

ServerId	Unique identifier of the server.
Name	Name of the server.
Status	Current state of the server.
StatusInfo	Current state of the server in a human readable form.

Operations

start()	Starts the server.
stop()	Stops the server.

18.1.2. Engine

ObjectName: "AgentServer:server=AgentServer#0,cons=Engine#0"⁸

Attributes

Running	True if the engine is active.
AgentProfiling	True if the profiling of agents is globally active. Profiling accumulates the execution and commit time for each agent.
NbAgents	The number of agents actually loaded in memory.
NbFixedAgents	The number of fixed agents (always in memory).
NbMaxAgents	The maximum number of agents loaded in memory.
NbReactions	The number of agent's reaction since last boot.

⁸ #0 refers to the unique identifier of server.

NbMessages	The number of messages posted to this engine since creation.
NbWaitingMessages	The number of waiting messages in this engine.
AverageLoad[1 5 15]	The load averages for the last minute, last 5 minutes and last 15 minutes.
ReactTime	The cumulative time of execution of agents (if profiling is activated).
CommitTime	The cumulative time spent in the transaction (if profiling is activated).

Operations

start()	Starts Sthe engine.
stop()	Stops the engine.
report()	If fr.dyade.aaa.agent.MsgTypesTracking properties is set, returns a report about the statistic distribution of messages types in engine's queue.
resetCommitTime()	Sets CommitTime to 0.
resetReactTime()	Sets ReactTime to 0.
resetTimer()	Sets CommitTime and ReactTime to 0.
dumpAgent()	Prints a human readable view of the specified agent. Parameter: unique identifier of agent.

18.1.3. Agent

Each agent implements a MBean interface.

ObjectName: "AgentServer:server=AgentServer#0,cons=Engine#0,agent=<AgentName>[<AgentId>]"⁸

Attributes

Name	Symbolic name of the agent.
AgentId	Unique identifier of the agent.
Fixed	True if the agent is always in memory.
AgentProfiling	True if the profiling of this agent is active. Profiling accumulates the execution and commit time for each agent.
ReactNb	Number of reactions since last boot.
ReactTime	The cumulative time of execution of this agent (if profiling is activated).
CommitTime	The cumulative time spent in the transaction for this agent (if profiling is activated).

Operations

delete()	Removes this agent.
resetCommitTime()	Sets CommitTime to 0.
resetReactTime()	Sets ReactTime to 0.
resetTimer()	Sets CommitTime and ReactTime to 0.

18.2. Transaction

There are several implementations of the transactional persistence component. We detail below the two main ones used in Joram.

18.2.1. NGTransaction

NGTransaction is an implementation of the transactional persistence component based on the OS file system. It implements a specific transactional log system for high performance.

ObjectName: "AgentServer:server=AgentServer#0,cons=Transaction"⁸

Attributes

StartTime	The start time of transaction module (ms sinc epoch).
Phase	The current phase of transaction module.
PhaseInfo	The current phase of transaction module in a human readable format.
CommitCount	The number of commit operation since starting up.
GarbageCount	The number of garbage operation since starting up.
GarbageRatio	The percentage of time take by garbage operations since starting up.
GarbageTime	The cumulated time of garbage operations.
NbLogFiles	The maximum number of rolled log files used by the component.
LogFileSize	The size of current disk log in Mb.
LogMemorySize	The number of operation in the memory log.
MaxLogFileSize	The maximum size of a disk log in Mb, by default 16Mb.
NbLoadedObjects	The number of load operation from repository.
NbLoadedFromLog	The number of load operation from a log file since last start.
NbDeletedObjects	The number of delete operation on repository.
NbBadDeletedObjects	The number of useless delete operation on repository.
NbSavedObjects	The number of save operation on repository.

RepositoryImpl	The implementation class of repository.
SyncOnWrite	True if every write in the log file is synced to disk.

Operations

logCounters()	The number of valid operation for each log.
logContent()	The list of operation for the specified log. Parameter: id of log file.
garbage()	Explicit activation of log garbage.

18.2.2. JDBC Transaction

JDBC Transaction is an implementation of the transactional persistence component based on the use of an external database. It uses the JDBC API to connect and interact with this database.

ObjectName: "AgentServer:server=AgentServer#0,cons=Transaction"⁸

Attributes

StartTime	The start time of transaction module (ms sinc epoch).
Phase	The current phase of transaction module.
PhaseInfo	The current phase of transaction module in a human readable format.
CommitCount	The number of commit operation since starting up.
CommitBytes	The cumulated number of bytes writes during commit.
ClientInfo	A list containing the name and current value of each client info property supported by the driver.
NbInserts	Number of insertion statements that the component has executed since the last boot (it corresponds to the creation of new objects).
BadInserts	Number of insertion statements rejected by the database since the last boot (existing objects that need updating).
NbUpdates	Number of update statements that the component has executed since the last boot (it corresponds to the update of existing objects).
BadUpdates	Number of update statements rejected by the database since the last boot (update non-existent objects to create).
NbDeletes	Number of delete statements that the component has executed since the last boot
Driver	The classname of JDBC driver used.
URL	The JDBC URL used to configure the driver.
PropertiesPath	The pathname of configuration properties if it exists.

User	The username used to connect to the database.
DBName	Name of the database used to build the JDBC URL if needed. By default, concatenation of "JoramDB" and server unique id.
DBTableName	Name of the database table used to store the objects. It is used to build default statements if needed. By default, concatenation of "JoramDB" and server unique id.
DBInitStatement	SQL statement allowing to create the table used by the module.
DBInsertStatement	SQL statement allowing to insert an entry in the table used by the module, by default: "INSERT INTO <DBTableName> VALUES (?, ?)"
DBUpdateStatement	SQL statement allowing to update an entry in the table used by the module, by default: "UPDATE <DBTableName> SET content=? WHERE name=?"
DBDeleteStatement	SQL statement allowing to delete an entry in the table used by the module, by default: "DELETE FROM <DBTableName> WHERE name=?"
DBCloseStatement	SQL statement executed at the end of the module, by default none.

Operations

getObject()	The number of valid operation for each log. Parameter1: Prefix of object. Parameter2: Name of object.
getObjectList()	The list of operation for the specified log. Parameter: Prefix of object.

18.3. JNDI

18.3.1. NamingContext

ObjectName: "JNDI:nc=<name>"

Attributes

NamingContext	The list of reference of this context.
---------------	--

Operations

lookup()	Returns a readable view of the Reference. Parameter: name of object,
unbind()	Removes the reference Parameter: name of object,
createSubcontext()	Creates a new subcontext. Parameter: name of subcontext,
destroySubcontext()	Removes the subcontext.

18.3.2. JNDI TCP connector

ObjectName: "JNDI:service=tcp"

Attributes

ListenPort	The listen port of the server.
PoolSize	The number of threads handling incoming tcp connections.

18.4. Joram#0

18.4.1. JMS Connection Manager

ObjectName: "Joram#0:type=Connection"⁸

Attributes

Activated	Tells if the ConnectionManager is active.
InitiatedConnectionCount	The number of initiated connections since server start.
RunningConnectionsCount	The number of living connections.
FailedLoginCount	The number of connections rejected due to a failed authentication.

Operations

activate()	Activates the connection manager, and recursively all existing connectors. Creation of new connections will be allowed.
closeAllConnections()	Closes all opened connections on all existing connectors.
deactivate()	Deactivates the connection manager, and recursively all existing connectors. No new connection will be opened.

18.4.2. JMS TCP/SSL Connector

ObjectName: "Joram#0:type=Connection,mode=tcp"⁸

Attributes

Activated	Tells if the Connector is active.
InitiatedConnectionCount	The number of initiated connections since server start.
RunningConnectionsCount	The number of living connections.
FailedLoginCount	The number of connections rejected due to a failed authentication.
TcpListenerPoolSize	The number of threads listening for incoming tcp connections.
ServerAddress	The socket address of the server.
ListenPort	The listen port of the server.
ProtocolErrorCount	The number of connections rejected due to a wrong protocol header.

Operations

activate()	Activates the connection manager. Creation of new connections will be allowed.
closeAllConnections()	Closes all opened connections.
deactivate()	Deactivates the connection manager. No new connection will be opened.

18.4.3. JMS Connection

ObjectName: "Joram#0:type=Connection,mode=tcp,id=<identifiant>"⁸

Attributes

CreationDate	The date of creation of the connection.
Address	The socket address of remote client.
UserName	The name of the connected user.
ReceivedCount	The number of packet received on the connection.
SentCount	The number of packet sent on the connection.

Operations

close()	Closes the connection.
---------	------------------------

18.4.4. Queue

ObjectName: "Joram#0:type=Destination,name=<queue name>"⁸

This Object inherits from Agent MBean (see 18.1.3).

Attributes

CreationDate	The human readable creation time of this destination.
CreationTimeInMillis	The creation time of this destination as a long.
Period	The delay between all periodic tasks of this destination (cleaning, etc.).
Threshold	The threshold value of this queue, -1 if not set.
DeliveryDelay	The delivery delay in milliseconds used to wait before delivering a message. If set the resulting delay is the max between this value and the message property.
RedeliveryDelay	The delay in seconds use to wait before re-delivering messages after a deny.
FreeReading	True if this destination is free for reading.
FreeWriting	True if this destination is free for writing.
NbMaxMsg	The maximum number of pending messages for the destination, -1 if the limit is unset.
NbMsgsReceiveSinceCreation	The number of messages received since creation time of this destination.
NbMsgsDeliverSinceCreation	The number of messages delivered since creation time of this destination. It includes messages all delivered messages to a consumer, already acknowledged or not.
NbMsgsDeniedSinceCreation	The number of messages denied since creation time of this destination.
DeliveredMessageCount	The number of messages delivered and waiting for acknowledge.
NbMsgsSentToDMQSinceCreation	The number of erroneous messages forwarded to the DMQ since creation time of this destination.
PendingMessageCount	The number of pending messages in the queue.
WaitingRequestCount	The number of waiting requests in the queue.
Pause	Ture if the delivery of messages is suspended.
ConsumerLoad	The average consumer's load during last moments.
ProducerLoad	The average producer's load during last moments.

Operations

cleanPendingRequest	Removes all request that the expiration time is expired.
cleanWaitingMessage	Removes all messages that the time-to-live is expired.

18.4.5. Topic

ObjectName: "Joram#0:type=Destination,name=<topic name>"⁸
 This Object inherits from Agent MBean (see 18.1.3).

Attributes

CreationDate	The human readable creation time of this destination.
CreationTimeInMillis	The creation time of this destination as a long.
Period	The delay between all periodic tasks of this destination (cleaning, etc.).
NbMsgsReceiveSinceCreation	The number of messages received since creation time of this destination.
NbMsgsDeliverSinceCreation	The number of messages delivered since creation time of this destination. It includes messages all delivered messages to a consumer, already acknowledged or not.
NumberOfSubscribers	Number of subscribing users.
FreeReading	True if this destination is free for reading.
FreeWriting	True if this destination is free for writing.

18.4.6. User

ObjectName: "Joram#0:type=User,name=<user name>"⁸
 This Object inherits from Agent MBean (see 18.1.3).

Attributes

NbMaxMsg	The maximum number of pending messages for each subscription, -1 if the limit is unset.
Period	The delay between all periodic tasks of this destination (cleaning, etc.).
RedeliveryDelay	The delay in seconds use to wait before re-delivering messages after a deny.
Threshold	The threshold value of this queue, -1 if not set.
NbMsgsSentToDMQSinceCreation	The number of erroneous messages forwarded to the DMQ since creation time of this destination.

18.4.7. JMSModule (bridge JMS)

ObjectName: "BridgeJMS#0:type=Connections,name=<connection name>"⁸

Attributes

State	The current status of Bridge module.
NamingFactory	The JNDI factory class to use.
NamingURL	The URL of JNDI naming service to use.
CnxFactName	The ConnectionFactory JNDI name.
ClientId	The ClientId to use for JMS connections.

19. Environment variables

19.1. Server-side

These properties are used by the server's JVM ..

```
"Engine", "fr.dyade.aaa.agent.Engine"
"Channel", "fr.dyade.aaa.agent.Channel"
Engine.threadPriority, Thread.MAX_PRIORITY
Engine.recoveryPolicy
NbMaxAgents, 1000
fr.dyade.aaa.agent.Message$Pool.size", 150
```

AgentProfiling, false

- * Boolean value indicating if the agent profiling is on, by default false.
- * If true, the cumulative time of reaction and commit is kept for each agent.
- * In addition the total reaction and commit time is calculated for this engine.
- * This value can be adjusted through the AgentProfiling system property.

```
 /**
 * Name of property allowing to track the distribution of the types of messages,
 * by default false. If true, for each type of messages the total number of messages
 * sent since the beginning and the number of those waiting is counted.
 * <p>
 * This property can be fixed either from <code>java</code> launching command or
 * a3servers.xml configuration file.
 */
public static final String MSG_TYPES_TRACKING = "fr.dyade.aaa.agent.MsgTypesTracking";
```

19.1.1. Configuration

XML configuration file

The two Java properties below allows to specify the path and filename of the configuration file.

fr.dyade.aaa.agent.A3CONF_DIR

Property allowing to configure the directory to search the XML server configuration, by default the working directory of the process. This property can only be fixed from java launching command.

Be careful, the XML server configuration file is normally used only for the initial starting of the server, the configuration is then atomically maintained in the persistence directory.

fr.dyade.aaa.agent.A3CONF_FILE = “a3servers.xml”

Property allowing to configure the filename of the XML server configuration, by default “a3servers.xml”. This property can only be fixed from java launching command.

Be careful, the XML server configuration file is normally used only for the initial starting of the server, the configuration is then atomically maintained in the persistence directory.

Default configuration

If there is no configuration at server launching a simple default configuration is generated to describe the starting server (0 in the example below) :

```
<config>
  <property name="Transaction" value="fr.dyade.aaa.ext.NGTransaction"/>

  <server id="0" name="S0" hostname="host">
    <service class="org.objectweb.joram.mom.proxies.ConnectionManager"
      args="root root"/>
    <service class="org.objectweb.joram.mom.proxies.tcp.TcpProxyService"
      args="16010"/>
    <service class="fr.dyade.aaa.jndi2.server.JndiServer"
      args="16400"/>
  </server>
</config>
```

The *id* attribute of the server description is the one of the launching server and the *hostname* attribute corresponds to the DNS name of the current host. Many of the parameters of this configuration can be controlled through a set of system properties (see below).

fr.dyade.aaa.agent.A3CONF_ADMINUID = “root”

Property allowing to configure the administrator user name when using the default server configuration, by default "root". This Configuration is automatically generated at first starting if no XML configuration file is found. This property can only be fixed from `<code>java</code>` launching command.

Be careful, this configuration is normally used only for the initial starting of the server, the configuration is then atomically maintained in the persistence directory.

fr.dyade.aaa.agent.A3CONF_ADMINPWD

Property allowing to configure the administrator password when using the default server configuration, by default it is the same that the administrator user name. This Configuration is automatically generated at first starting if no XML configuration file is found. This property can only be fixed from `<code>java</code>` launching command.

Be careful, this configuration is normally used only for the initial starting of the server, the configuration is then atomically maintained in the persistence directory.

fr.dyade.aaa.agent.A3CONF_JMS_PORT = “16010”

Property allowing to configure the listening port of the JMS server when using the default server configuration, by default 16010. This Configuration is automatically generated at first starting if no XML configuration file is found. This property can only be fixed from `<code>java</code>` launching command.

Be careful, this configuration is normally used only for the initial starting of the server, the configuration is then atomically maintained in the persistence directory.

fr.dyade.aaa.agent.A3CONF_JNDI_PORT = "16400"

Property allowing to configure the listening port of the JNDI server when using the default server configuration, by default 16400. This Configuration is automatically generated at first starting if no XML configuration file is found. This property can only be fixed from <code>java</code> launching command.

Be careful, this configuration is normally used only for the initial starting of the server, the configuration is then atomically maintained in the persistence directory.

19.1.2. OSGi

These properties are used when Joram is launched in an OSGi environment as a set of bundles.

fr.dyade.aaa.agent.AgentServer.id

Property allowing to configure the server identifier, by default 0. This property can be fixed either from the OSGi configuration file, or as a system property.

fr.dyade.aaa.agent.AgentServer.storage

Property allowing to configure the path of the persistent storage, by default the directory named "s0". This property can be fixed either from the OSGi configuration file, or as a system property.

fr.dyade.aaa.osgi.exit

Property allowing to configure the behavior of OSGi framework in case of failure, by default false. This property can be fixed either from the OSGi configuration file, or as a system property.

If true the OSGi framework is stopped if there is an error during AgentServer starting, or if the a3 bundle is stopped.

19.1.3. Server Health checking

fr.dyade.aaa.agent.exitOnServiceFailure = "false"

Property specifying that the server should stop if any of the services doesn't start correctly. By default false. This property can be fixed either from XML configuration file or Java launching command.

fr.dyade.aaa.agent.check.period = "-1"

Property allowing to configure the checking period of the server health (in ms). The corresponding value is the period of time beyond which an error is thrown and the registered listeners are called. The activation period is 5 times lower.

The task logs warning and errors if abnormal behavior are detected, the final behavior is depending of the registered listeners.

The default value is -1, if the value is less than 0 the checking is not activated. This value cannot be less than 10s. This property can be fixed either from XML configuration file or Java launching command.

19.1.4. File transacted persistence

The persistence module implementation is specified by the Transaction property. There are multiple implementation classes, each defines its own set of properties.

⚠ Warning : Be careful, all these properties below are needed to initialize the persistence module. Therefore, they can not be defined in the a3servers.xml configuration file that is stored in persistence after the first boot.

NullTransaction

The implementation class of the NullTransaction module is "fr.dyade.aaa.util.NullTransaction". This implementation is special because it does not implement persistence, at each boot the server starts in its initial state. There is no property allowing to configure this simple implementation.

NTransaction

The implementation class of the NTransaction module is "fr.dyade.aaa.util.NTransaction". This implementation has been during a long time the reference implementation, the NGTransaction module offers now best performances.

NTLogMemorySize = "2048"

This property allows to define in Kb the maximum size of memory log, by default 2048Kb.
This property can be set only at first launching.

NTLogFileSize = "16"

This property allows to define in Mb the maximum size of disk log, by default 16Mb.
This property can be set only at first launching.

Transaction.SyncOnWrite = "false"

This property allows to define if every write in the log file is synced to disk, by default false.
This property can be set only at first launching.

Transaction.UseLockFile = "true"

This property allows to define if a lock file is used to avoid multiples activation of the Transaction component.
This property can be set only at first launching.

NTLogMemoryCapacity = "4096"

This property allows to define the capacity of the memory log, by default 4096.
This property can be set only at first launching.

NTLogThresholdOperation = "1000"

This property allows to define the number of pooled operation, by default 1000.
This property can be set only at first launching.

NGTransaction

The implementation class of the NGTransaction module is "fr.dyade.aaa.ext.NGTransaction".

Transaction.MaxLogFileSize = "16"

This property allows to define the maximum size of each disk log in Mb, by default 16Mb.
This property can be set only at first launching.

Transaction.NbLogFile = "4"

This property allows to define the maximum number of disk log used by the Transaction component, by default 4.
This property can be set only at first launching.

Transaction.minObjInLog = "64"

This property allows to define the minimum number of 'live' objects in a disk log before a garbage, by default 64.
This property can be set only at first launching.

Transaction.SyncOnWrite = "false"

This property allows to define if every write in the log file is synced to disk, by default false.
This property can be set only at first launching.

Transaction.RepositoryImpl = "fr.dyade.aaa.util.FileRepository"

This property allows to define the Repository class name implementation. By default the FileRepository stores each persistent object in files, alternatively there are Repository implementations for storing the objects in a database. This property can be set only at first launching.

Transaction.UseLockFile = "true"

This property allows to define if a lock file is used to avoid multiples activation of the Transaction component. This property can be set only at first launching.

Transaction.LogMemoryCapacity = "4096"

This property allows to define the capacity of the memory log, by default 4096. This property can be set only at first launching.

Transaction.LogThresholdOperation = "1000"

This property allows to define the number of pooled operation, by default 1000. This property can be set only at first launching.

19.1.5. Database transacted persistence

The persistence module implementation is specified by the Transaction property. There are multiple implementation classes based on Database persistence, each inherits from DBTransaction but defines its own set of properties.

✖ Warning : Be careful, all these properties below are needed to initialize the persistence module. Therefore, they can not be defined in the a3servers.xml configuration file that is stored in persistence after the first boot. Any changes after the first boot can cause unpredictable behavior..

org.ow2.joram.dbtransaction.dbtable = "JoramDB"

This property allows to define the name of the table used by the persistence module, by default "JoramDB". This property can be set only at first launching.

MySQLDBTransaction

The implementation class of the MySQLDBTransaction module is "fr.dyade.aaa.util.MySQLDBTransaction", this module is used to implement the persistence through JDBC to a MySQL database.

The configuration is done through a file of properties; its path name is given by the "MySQLDBTransactionConfigFile" property, by default "MySQL.properties". This file defines all properties used to configure the module:

- The URL used by the SQL drive is "jdbc:mysql://host:port/dababase" where "host", "port" and "database" are properties.
- "user" and "password" properties define authentication.
- All additional properties are added at the end of the URL as parameters with the syntax "&name=value".

✖ Warning : This module is now deprecated and should be replaced by JDBCTransaction that implements a generic Transaction component on top of JDBC.

DerbyDBTransaction

The implementation class of the DerbyDBTransaction module is "fr.dyade.aaa.util.DerbyDBTransaction", this module is used to implement the persistence through JDBC to an embedded Derby database.

The module creates a private Derby database with no configuration parameter.

✖ Warning : This module is now deprecated and should be replaced by JDBCTransaction that implements a generic Transaction component on top of JDBC.

JDBCTransaction

The implementation class of the JDBCTransaction module is "fr.dyade.aaa.util.JDBCTransaction", this module is used to implement the persistence in a database through a JDBC driver.

org.ow2.joram.jdbc.transaction.driver

This property allows to define the class that implements the JDBC driver, for example "com.mysql.jdbc.Driver". This class needs to be in the classpath of the server.

org.ow2.joram.jdbc.transaction.url

This property allows to define the database url of the form "jdbc:subprotocol://host:port/dbname". If this property is defined, the properties "protocol", "host", "port" and "dbname" below are ignored.

org.ow2.joram.jdbc.transaction.protocol

This property allows to define the protocol part of the URL used to establish the connection, for example "jdbc:mysql". It is ignored if the url property is defined.

org.ow2.joram.jdbc.transaction.host

This property allows to define the hostname of the URL used to establish the connection. It is ignored if the url property is defined.

org.ow2.joram.jdbc.transaction.port

This property allows to define the port of the URL used to establish the connection. It is ignored if the url property is defined.

org.ow2.joram.jdbc.transaction dbname

This property allows to define the database name of the URL used to establish the connection, by default concatenation of "JoramDB" and server unique id. It is ignored if the url property is defined.

org.ow2.joram.jdbc.transaction.properties

This property allows to define the name of a file containing a list of arbitrary string tag/value pairs as connection arguments; normally at least a "user" and "password" property should be included.

org.ow2.joram.jdbc.transaction.user

This property allows to define the name of database user on whose behalf the connection is being made. If this value is already set in properties it is ignored.

org.ow2.joram.jdbc.transaction.password

This property allows to define the password of database user on whose behalf the connection is being made. If this value is already set in properties it is ignored.

org.ow2.joram.dbtransaction.dbtable

This property allows to define the name of the table used to store objects in database, by default concatenation of "JoramDB" and server unique id. It is used to build all default statements.

org.ow2.joram.jdbc.transaction.dbinit

This property allows to define the SQL statement allowing to create the table used by the module, for example: "CREATE TABLE JoramDB (name VARCHAR(256), content LONG VARCHAR FOR BIT DATA, PRIMARY KEY(name))"

This property is required, there is no default value.

org.ow2.joram.jdbc.transaction.dbinsert

This property allows to define the SQL statement allowing to insert an entry in the table used by the module, by default: "INSERT INTO <>VALUES (?, ?)".

org.ow2.joram.jdbc.transaction.dbupdate

This property allows to define the SQL statement allowing to update an entry in the table used by the module, by default: "UPDATE <>table<> SET content=? WHERE name=?".

org.ow2.joram.jdbc.transaction.dbload

This property allows to define the SQL statement allowing to load an entry from the table used by the module, by default: "SELECT content FROM <>table<> WHERE name=?".

org.ow2.joram.jdbc.transaction.dbdelete

This property allows to define the SQL statement allowing to delete an entry in the table used by the module, by default: "DELETE FROM <>table<> WHERE name=?".

org.ow2.joram.jdbc.transaction.dbclose

This property is used to define the SQL statement executed at the end of the module, by default none.

org.ow2.joram.jdbc.transaction.connect_retry_count

This property is used to set the number of reconnection attempts after a failure, by default 5. If set to 0, the number of retries is not limited.

org.ow2.joram.jdbc.transaction.connect_retry_min_delay

This property is used to set the minimum time between 2 attempts to reconnect after a failure, by default 1.000 (1 seconds). If set to 0, there is no delay..

org.ow2.joram.jdbc.transaction.connect_retry_max_period

This property is used to set the maximum time trying to reconnect after a failure, by default 60.000 (60 seconds). If set to 0, the trial period is unlimited.

19.1.6. Communication components

Network

This Class defines the basic behavior of all network components, its properties are valid for all network components.

WDActivationPeriod = "1000L"

This property allows to define the period of time in millisecond between two activations of watch-dog thread, default value is 1000L (1 second). This value can be adjusted for all Network components by setting a global WDActivationPeriod property or for a particular network by setting dname.WDActivationPeriod specific property (where dname is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in a3servers.xml configuration file.

WDNbRetryLevel1 = "5"

This property allows to define the number of connection attempt at stage 1, default value is 5. This value can be adjusted for all Network components by setting a global WDNbRetryLevel1 property or for a particular network by setting dname.WDNbRetryLevel1 specific property (where dname is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in a3servers.xml configuration file.

WDRetryPeriod1

This property allows to define the period of time in ms between two connection try at stage 1, default value is WDActivationPeriod. Be careful, in most Network components setting this value to a value less than

WDActivationPeriod is useless. In the same way, the real waiting period between two connection attempts is depending of the connection timeout.

This value can be adjusted for all Network components by setting a global *WDRetryPeriod1* property or for a particular network by setting *dname.WDRetryPeriod1* specific property (where *dname* is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in *a3servers.xml* configuration file.

WDNbRetryLevel2

This property allows to define the number of try at stage 2, default value is 30.

This value can be adjusted for all Network components by setting a global *WDNbRetryLevel2* property or for a particular network by setting *dname.WDNbRetryLevel2* specific property (where *dname* is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in *a3servers.xml* configuration file.

WDRetryPeriod2

This property allows to define the period of time in ms between two connection try at stage 2, default value is 10000L (10 seconds). Be careful, in most Network components setting this value to a value less than *WDActivationPeriod* is useless. In the same way, the real waiting period between two connection attempts is depending of the connection timeout.

This value can be adjusted for all Network components by setting a global *WDRetryPeriod2* property or for a particular network by setting *dname.WDRetryPeriod2* specific property (where *dname* is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in *a3servers.xml* configuration file.

WDRetryPeriod3

This property allows to define the period of time in ms between two connection try at stage 3, default value is 60000L (1 minute). Be careful, in most Network components setting this value to a value less than *WDActivationPeriod* is useless. In the same way, the real waiting period between two connection attempts is depending of the connection timeout.

This value can be adjusted for all Network components by setting a global *WDRetryPeriod3* property or for a particular network by setting *dname.WDRetryPeriod3* specific property (where *dname* is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in *a3servers.xml* configuration file.

StreamNetwork

This Class specializes the behavior of the Network class for a TCP based component. It inherits all properties defines by the Network class.

CnxRetry

This property allows to define the number of attempts to bind the server's socket before aborting, default value is 3.

This value can be adjusted for all Network components by setting a global *CnxRetry* property or for a particular network by setting *dname.CnxRetry* specific property (where *dname* is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in *a3servers.xml* configuration file.

backlog

This property allows to define the maximum queue length for incoming connection indications, default value is 5.

This value can be adjusted for all Network components by setting a global *backlog* property or for a particular network by setting *dname.backlog* specific property (where *dname* is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in *a3servers.xml* configuration file.

TcpNoDelay

This property allows to enable/disable TCP Nagle's algorithm for the related TCP connections, default value is false.

This value can be adjusted for all Network components by setting a global *TcpNoDelay* property or for a particular network by setting *dname.TcpNoDelay* specific property (where *dname* is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in *a3servers.xml* configuration file.

SoLinger

This property allows to enable the TCP SO_LINGER property with the specified linger time in seconds, if the value is less than 0 then it disables SO_LINGER. Default value is -1.

This value can be adjusted for all Network components by setting a global *SoLinger* property or for a particular network by setting *dname.SoLinger* specific property (where dname is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in a3servers.xml configuration file.

SoTimeout

This property allows to enable/disable SO_TIMEOUT with the specified timeout in milliseconds. The timeout must be > 0. A timeout of zero is interpreted as an infinite timeout. Default value is 0.

This value can be adjusted for all Network components by setting a global *SoTimeout* property or for a particular network by setting *dname.SoTimeout* specific property (where dname is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in a3servers.xml configuration file.

ConnectTimeout

This property allows to define in milliseconds the timeout used during socket connection. The timeout must be > 0. A timeout of zero is interpreted as an infinite timeout. Default value is 0.

This value can be adjusted for all Network components by setting a global *ConnectTimeout* property or for a particular network by setting *dname.ConnectTimeout* specific property (where dname is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in a3servers.xml configuration file.

InLocalAddress

This property allows to define the local address the listen ServerSocket is bound to. A null address will assign the wildcard address. Default value is null.

This value can be adjusted for all Network components by setting a global *InLocalAddress* property or for a particular network by setting *dname.InLocalAddress* specific property (where dname is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in a3servers.xml configuration file.

OutLocalPort

This property allows to define the local port the sockets are bound to. A valid port value is between 0 and 65535. A port number of zero will let the system pick up an ephemeral port in a bind operation. Default value is 0.

This value can be adjusted for all Network components by setting a global *OutLocalPort* property or for a particular network by setting *dname.OutLocalPort* specific property (where dname is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in a3servers.xml configuration file.

OutLocalAddress

This property allows to define the local address the sockets are bound to. A null address will assign the wildcard address. Default value is null.

This value can be adjusted for all Network components by setting a global *OutLocalAddress* property or for a particular network by setting *dname.OutLocalAddress* specific property (where dname is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in a3servers.xml configuration file.

ServerSocketFactory

This property allows to define a specific factory for ServerSocket in order to by-pass compatibility problem between JDK version. Currently there is two factories, The default factory one for JDK since 1.4, and ServerSocketFactory13 for JDK prior to 1.4.

This value can be adjusted for all Network components by setting a global *ServerSocketFactory* property or for a particular network by setting *dname.ServerSocketFactory* specific property (where dname is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in a3servers.xml configuration file.

SocketFactory

This property allows to define a specific factory for Socket in order to by-pass compatibility problem between JDK version. Currently there is two factories, The default factory one for JDK since 1.4, and link SocketFactory13 for JDK prior to 1.4.

This value can be adjusted for all Network components by setting a global *SocketFactory* property or for a particular network by setting *dname.SocketFactory* specific property (where dname is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in a3servers.xml configuration file.

SimpleNetwork

The SimpleNetwork class is a simple implementation of TCP based network component with a single connection at a time, it inherits all properties defines from StreamNetwork and Network classes.

PoolNetwork

The PoolNetwork class is an efficient implementation of TCP based network component with multiples connections at a time, it inherits all properties defines from StreamNetwork and Network classes.

PoolNetwork.nbMaxCnx

This property defines the maximum number of concurrent connected sessions. By default this property is set to -1 to dynamically adjust to the number of servers of the domain (excepting the current server). Setting this value needs precautions to avoid unexpected connection loss.

This value can be adjusted for all Network components by setting a global *PoolNetwork.nbMaxCnx* property or for a particular network by setting *dname.nbMaxCnx* specific property (where dname is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in a3servers.xml configuration file.

PoolNetwork.IdleTimeout

This property defines in milliseconds the maximum idle period permitted before reseting the connection. The timeout must be > 0. A timeout of zero is interpreted as an infinite timeout. Default value is 60000 (1 minute), value less than 1000 are unauthorized.

This value can be adjusted for all Network components by setting a global *PoolNetwork.IdleTimeout* property or for a particular network by setting *dname.IdleTimeout* specific property (where dname is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in a3servers.xml configuration file.

PoolNetwork.compressedFlows

This property defines if the streams between servers are compressed or not. Default value is false, be careful in a specific domain all servers must use the same definition.

This value can be adjusted for all Network components by setting a global *PoolNetwork.compressedFlows* property or for a particular network by setting *dname.compressedFlows* specific property (where dname is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in a3servers.xml configuration file.

PoolNetwork.maxMessageInFlow

This property defines the default value for maximum number of message sent and non acknowledged on a connection. By default this value is set to -1 and there is no flow control.

This value can be adjusted for all Network components by setting a global *PoolNetwork.maxMessageInFlow* property or for a particular network by setting *dname.maxMessageInFlow* specific property (where dname is the name of the specific domain).

For a particular network the value can be defined finely for a the connection with a particular remote server #N by setting *PoolNetwork.maxMessageInFlow_N* or *dname.maxMessageInFlow_N*.

Theses properties can be fixed either from Java launching command, or in a3servers.xml configuration file.

SSLNetwork

The SSLNetwork class is a secure version of the PoolNetwork component using SSL sockets, it inherits all properties defines from PoolNetwork class.

`KeyManagerFactory.getInstance("SunX509");`

HttpsNetwork.keyfile = ".keystore"

This property allows to fix the keystore's pathname. By default the key file is ".keystore".

This property can be fixed either from Java launching command, or in a3servers.xml configuration file.

HttpsNetwork.pass = "changeit"

This property allows to fix the keystore's password. By default the password is "changeit".

This property can be fixed either from Java launching command, or in a3servers.xml configuration file.

fr.dyade.aaa.agent.SSLNetwork.SSLContext = "TLS"

To be completed

fr.dyade.aaa.agent.SSLNetwork.KeyStoreType = "JKS"

To be completed

HttpNetwork

The HttpNetwork class is an implementation of network component based on HTTP 1.1 protocol, it inherits all properties defines from StreamNetwork and Network classes.

proxyhost

These properties allows to define the hostname (or IP dotted address) of proxy host, if not defined there is a direct connection between the client and the server. This value can be adjusted for all HttpNetwork components by setting a global proxyhost property or for a particular network by setting dname.proxyhost specific property (where dname is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in a3servers.xml configuration file.

proxypot

These properties allows to define the port number of proxy if any. This value can be adjusted for all HttpNetwork components by setting proxypot global property or for a particular network by setting dname.proxypot specific property (where dname is the name of the specific domain). By default the port 8080 is used.

Theses properties can be fixed either from Java launching command, or in a3servers.xml configuration file.

ActivationPeriod = "10000"

These properties allows to define the period of time between two activation of NetServerOut, it matches to the time between two requests from the client to the server when there is no message to transmit from client to server. This value can be adjusted for all HttpNetwork components by setting ActivationPeriod global property or for a particular network by setting dname.ActivationPeriod specific property (where dname is the name of the specific domain). By default, its value is 10000 (10 seconds).

Theses properties can be fixed either from Java launching command, or in a3servers.xml configuration file.

NbDaemon

These properties allows to define the number of listening daemon, this value is only valid for the server part of the HttpNetwork. This value can be adjusted for all HttpNetwork components by setting NbDaemon global property or for a particular network by setting dname.NbDaemon specific property (where dname is the name of the specific domain).

Theses properties can be fixed either from Java launching command, or in a3servers.xml configuration file.

HttpsNetwork

The HttpsNetwork class is a secure version of the HttpNetwork component based on HTTPS protocol, it inherits all properties defines from HttpNetwork class.

```
KeyStore.getInstance("JKS");
KeyManagerFactory.getInstance("SunX509");
SSLContext.getInstance("TLS");
```

HttpsNetwork.keyfile = “.keystore”

This property allows to fix the keystore's pathname. By default the key file is “.keystore”.

This property can be fixed either from Java launching command, or in a3servers.xml configuration file.

HttpsNetwork.pass = “changeit”

This property allows to fix the keystore's password. By default the password is “changeit”.

This property can be fixed either from Java launching command, or in a3servers.xml configuration file.

UDPNetwork**UDPReceiveBufferSize = “1048576”****UDPSendBufferSize = “8192”*****19.1.7. Joram service*****org.objectweb.joram.tempQ.dmqOnDel = “false”**

This property is used to define the behavior when a temporary queue is destroyed (explicitly or automatically due to a connection closure). By default, the messages in the queue are destroyed along with the queue. If this property is set to true, the remaining messages are sent to the DMQ if it is configured.

org.objectweb.joram.mom.dest.strictCounters = “false”

To meet performance goals, counters in destinations can give a consistent but out-of-date view after a server crash. If this property is set to true, the destination's counters are handled in transactions in the same way as functional datas. So the counters will always be up-to-date, even after a server crash.

org.objectweb.joram.mom.messages.SWAPALLOWED = “false”

This property allows to define if the swapping mechanism is globally activated for messages in this server, default value is false.

Note: the message swapping can be finely configured for each message using the JMS_JORAM_SWAPALLOWED property of the JMS message.

This property can be fixed either from Java launching command, or in a3servers.xml configuration file.

org.objectweb.joram.mom.messages.USELOADALL = “false”

This property allows to define if the queue restoration at startup must use the loadAll transaction mechanism when it is implemented. This mechanism can improve performance with database transacted persistence, especially when a large number of messages are present in the queue at startup.

This property can be fixed either from Java launching command, or in a3servers.xml configuration file. Default value is false.

org.objectweb.joram.mom.proxies.ConnectionManager.CtrlFlowThreshold

Property allowing to define the threshold beyond which the flow-control of incoming messages is activated. Default value is 25. This mechanism slowed the flow of incoming messages when the system load exceeds this threshold.

This property can be fixed either from Java launching command, or in a3servers.xml configuration file.

org.objectweb.joram.mom.proxies.ConnectionManager.CtrlFlowThroughput = "100000"

Property allowing to define the average throughput (message per seconds) of the server for the calculation of flow control, default value is 100000.

This property can be fixed either from Java launching command, or in a3servers.xml configuration file.

org.objectweb.joram.mom.proxies.ConnectionManager.multiCnxSync = "false"

Property allowing to activate the synchronization mode of multiples connections, by default false. This mode allows to pack commands that occurs in a same time in order to minimize the number of transactions.

This property can be fixed either from Java launching command, or in a3servers.xml configuration file.

org.objectweb.joram.mom.proxies.ConnectionManager.multiCnxSyncDelay = "1"

Property allowing to define in milliseconds the duration of instant to pack the commands, by default 1ms.

This property can be fixed either from Java launching command, or in a3servers.xml configuration file.

ConnectionManager.inFlow = "-1"

This property allows to define the limit throughput of incoming messages flow for this server (message per seconds), default value is -1 (no limitation).

This property can be fixed either from Java launching command, or in a3servers.xml configuration file.

TCP and SSL connection service

This service manages the TCP connectivity to the Joram server.

org.objectweb.joram.TcpConnection.ClockSynchro.Threshold = "1000"

Property allowing to change the threshold of warning for the verification of the synchronization between the client and server clock. A warning is generated if there is more than this value in milliseconds between the two clocks. By default the value is 1000 milliseconds.

org.objectweb.joram.mom.proxies.tcp.poolSize = "1"

This property allows to fix the pool size for the connection's listener, default value is 1.

This property can be fixed either from Java launching command, or in a3servers.xml configuration file.

org.objectweb.joram.mom.proxies.tcp.soTimeout = "5000"

This property allows to fix the TCP SO_TIMEOUT property during the client's connections, default value is 5.000 ms. It allows to avoid pending connection from non Joram client (telnet, etc.).

This property can be fixed either from Java launching command, or in a3servers.xml configuration file.

org.objectweb.joram.mom.proxies.tcp.backlog = "10"

This property allows to fix the TCP BACKLOG property for the client's connections, default value is 10.

This property can be fixed either from Java launching command, or in a3servers.xml configuration file.

org.objectweb.joram.TcpConnection.connectingTimer = "5000"

This property allows to fix the value of timeout during TCP connection, by default the value is 5000 milliseconds. If the server fails to establish the JMS connection during this time it closes it.

fr.dyade.aaa.util.ReliableTcpConnection.windowSize = "100"

19.1.8. JNDI Service

JNDIServer, DistributedJNDIServer

This service manages the TCP connectivity to the naming server. There are two implementations, a centralized one with a unique instance of the service (JNDIServer class) and a distributed that handles the synchronization between multiples instances of the server (DistributedJNDIServer class).

fr.dyade.aaa.jndi2.server.poolSize = "3"

This property allows to set the number of listening thread, default value is 3.

This property can be fixed either from Java launching command, or in a3servers.xml configuration file.

fr.dyade.aaa.jndi2.server.soTimeout = "10000"

This property is deprecated, use "fr.dyade.aaa.jndi2.socketTimeOut" instead.

fr.dyade.aaa.jndi2.socketTimeOut="5000"

This property allows to Enable/disable SO_TIMEOUT with the specified timeout in milliseconds, default value is 5.000L (5 seconds).

This property can be fixed either from Java launching command, or in a3servers.xml configuration file.

fr.dyade.aaa.jndi2.impl.LooseCoupling = "false"

This property sets the synchronization behavior of a distributed JNDI server, if true it sets the synchronization mode weakly coupled. By default its value is false.

This property can be fixed either from Java launching command, or in a3servers.xml configuration file.

19.1.9. Tools

AdminProxy

A AdminProxy service provides a TCP service allowing remote administration of agent servers through telnet for example. The TCP port number can be configured through the service parameter, by default this port is 8091.

fr.dyade.aaa.agent.AdminProxy.port = "8091"

This property allows to define the listen port for the AdminProxy service, by default 8091.

fr.dyade.aaa.agent.AdminProxy.nbm = "1"

This property allows to define the number of monitor waiting to handle requests, by default only one server thread is allocated.

19.2. Client-side

These properties are used by the client's JVM.

19.2.1. ConnectionFactory

These properties are used when creating and configuring ConnectionFactory.

JoramDfltRootLogin = "root"

Property allowing to change the default administrator login name for connection, normally the administrator login name is given by the client and this property is not used.
By default the administrator login name is "root".

JoramDfltRootPassword = "root"

Property allowing to change the default administrator login password for connection, normally the administrator login password is given by the client and this property is not used.
By default the administrator login password is "root".

JoramDfltLogin = "anonymous"

Property allowing to change the default login name for connection, normally the login name is given by the client and this property is not used.
By default the default login's name is "anonymous".

JoramDfltPassword = "anonymous"

Property allowing to change the default login password for connection, normally the login password is given by the client and this property is not used.
By default the default login's password is "anonymous".

JoramDfltServerHost = "localhost"

Property allowing to change the default server's hostname for connection, normally the server's hostname is defined in the connection factory and this property is not used.
By default the server's hostname is "localhost".

JoramDfltServerPort = 16010

Property allowing to change the default server's port for connection, normally the server's port is defined in the connection factory and this property is not used.
By default the port is 16010.

TCP's Connections

These properties are used when establishing TCP's (and SSL's) connections to the Joram server.

org.objectweb.joram.TcpConnection.ClockSynchro.Threshold = 1000

Property allowing to change the threshold of warning for the verification of the synchronization between the client and server clock. A warning is generated if there is more than this value in milliseconds between the two clocks.
By default the value is 1000 milliseconds.

org.objectweb.joram.client.jms.admin.requestTimeout = 60000

Property allowing to set the timeout before aborting an administration request.

SSL's Connections

These properties are used to establish SSL's connections to the Joram server. The SSL connector inherits properties and behavior from TCP connector.

org.objectweb.joram.keystore = "./joram_ks"

Property allowing to define the pathname of the keystore file where you have stored the certificate to be loaded.
By default the keystore's pathname is "./joram_ks".

org.objectweb.joram.keystorepass = "jorampass"

Property allowing to change the password associated with the keystore file.
By default the keystore password is "jorampass".

org.objectweb.joram.keystoretype = "JKS"

Property allowing to change the type of the keystore file.
By default the keystore type is "JKS".

org.objectweb.joram.sslCtx = "SSL"

Property allowing to select the standard name of the requested protocol.
By default the protocol used is "SSL".

org.objectweb.joram.clientAuth = "NEED"

Property allowing to control whether accepted server-mode SSLSockets will be initially configured to require or not client authentication.

A socket's client authentication setting is one of the following:

- WANT: client authentication required.
- NEED: client authentication requested.
- NONE: no client authentication desired.

The value of this parameter can be overloaded for new connections using the corresponding MBean.

19.2.2. JMS MessageID

Classically a JMS MessageID has the following structure "**ID:<proxyid>c<key>m<order>**" where <proxyid> identifies the user, <key> is the identifier of the connection and <order> is the serial number of the message in this connection. A mechanism allows to add a user prefix to this identifier then the JMS MessageID has the following structure "**ID:<prefix>_<proxyid>c<key>m<order>**".

Be careful, the JMS MessageID is part of the JMS message header, therefore its size has an impact on the volume of data transferred and stored on the server.

This prefix can be defined through the ConnectionFactory using either the programmatic interface, or the XML configuration scripts, or by the Java environment variable below.

org.objectweb.joram.client.jms.messageid.prefix

This property allows to customize the JMS MessageID adding the specified string. If the property is set to <prefix> then the JMS MessageID has the following structure "**ID:<prefix>_<proxyid>c<key>m<order>**".

This property overloads the definition of the corresponding property in the ConnectionFactory parameters.

19.2.3. JNDI

scn.naming.provider.url

Name of the property which defines the URL of the server when creating an initial context using this factory. If it is not defined the property *java.naming.provider.url* is searched. If it is not specified, the default configuration is determined by the service provider using "host" and "port" properties below.

scn.naming.factory.host

Name of the property which defines the host name of the server when creating an initial context using this factory. If it is not defined the property *java.naming.factory.host* is searched, by default the host "localhost" is used.

scn.naming.factory.port

Name of the property which defines the listener port used when creating an initial context using this factory. If it is not defined the property *java.naming.factory.port* is searched, by default the port 16400 is used.

fr.dyade.aaa.jndi2.client.SocketFactory = null;

This property allows to configure the use of a different socket factory for JNDI connections. Currently there is two factories, the default factory one for JDK since 1.4 (SocketFactory14), and SocketFactory13 for JDK prior to 1.4 :

- "fr.dyade.aaa.common.net.SocketFactory14"
- "fr.dyade.aaa.common.net.SocketFactory13"

Default value is "fr.dyade.aaa.common.net.SocketFactory14".

fr.dyade.aaa.jndi2.client.ConnectTimeout = 0

This property allows to define in milliseconds the timeout used during socket connection, this timeout must be greater or equal to 0, a timeout of zero is interpreted as an infinite timeout.
Default value is 0.

fr.dyade.aaa.jndi2.socketTimeOut="5000"

This property allows to Enable/disable SO_TIMEOUT with the specified timeout in milliseconds, default value is 5.000L (5 seconds).

fr.dyade.aaa.jndi2.socketLinger="-1"

This property allows to Enable/disable SO_LINGER with the specified value in seconds, default value is -1 (disable SO_LINGER).

20. Logging

20.1. Configuration

Joram defines numerous logging topics that allow to finely filter traces. Normally there is a topic defined for each Java class, this topic corresponds to the full name of the class. However, some topics can be used to log specific system functions, or sets of components. They are described below.

The user can enable 5 levels of logging: FATAL, ERROR, WARN, INFO and DEBUG.

- The FATAL level shows only very severe error events that will presumably lead the application to abort.
- The ERROR level shows all error events, even those that might still allow the application to continue running.
- The WARN level shows events corresponding to potentially abnormal situations.
- The INFO level shows informational messages that highlight the progress of the application at coarse-grained level. This level can lead to significant volumes of traces.
- The DEBUG Level shows fine-grained informational events that are most useful to debug an application.. This level generates large volumes of traces.

20.2. Topics

20.2.1. A3 – Agent runtime

The topics below are the main ones to observe the agent runtime.

- fr.dyade.aaa.agent.Agent: This topic allows to show the logging of every A3 Agent component.
- fr.dyade.aaa.agent.AgentServer: This topic allows to show the logging of the AgentServer component.
- fr.dyade.aaa.agent.Engine: This topic allows to show the logging of every A3 Engine component.
- fr.dyade.aaa.agent.Network: This topic allows to show the logging of every A3 Network component.
- fr.dyade.aaa.util.Transaction: This topic allows to show the logging of every A3 Transaction component.

20.2.2. Joram – JMS runtime

The topics below are the main ones to observe the JMS runtime.

- org.objectweb.joram: This topic allows to show the logging of all Joram's component, either client and server side.
- org.objectweb.joram.mom: This topic allows to show the logging of Joram's server components.
- org.objectweb.joram.mom.dest: This topic allows to focus on the traces of operation of the destinations (queues or topics).
- org.objectweb.joram.mom.proxies.UserAgent: This topic allows to focus on the traces of operation of the user proxies.
- org.objectweb.joram.client: This topic allows to show the logging of Joram's client components.

JMS Message tracking

The topics below help track the sending and consumption of messages. This makes it possible, among other things, to ensure that each message sent is well consumed.

- org.objectweb.joram.client.jms.Session.Message: Allows the tracking of messages on the client side.
- org.objectweb.joram.mom.dest.Queue.Message: Allows the tracking of messages on the server side.

JMS Connection and Session tracking

The topics below allow you to track the creation and closing of connections and sessions. The message is accompanied by a stacktrace that determines the origin of the event.

- org.objectweb.joram.client.jms.Connection.tracker: Track creation and closure of Connection.
- org.objectweb.joram.client.jms.Session.tracker: Track creation and closure of Session.

20.3. Error's messages server side

20.3.1. Initialisation / Starting

"NTransaction, cannot initializes the repository"

Exception donnée

"NTransaction.init(): Either the server is already running, either you have to remove lock file "
nom du fichier à supprimer donné

"can't instantiate transaction manager"

La classe du module transactionnelle est soit erronée, soit indisponible.

"can't start transaction manager"

Le module de persistance ne peut démarrer (cf. erreur correspondante)

"can't initialize services" ou "can't start services"

Un des services de la plateforme n'a pas pu démarrer

"problem during \$\$ starting"

Un des composants de la plateforme n'a pas pu démarrer : Engine et/ou Network

"Can't initialize AgentId, bad classpath"

Une des classes du serveur est manquante

"Can't initialize AgentId, storage problems"

L'initialisation du serveur est impossible du a des problèmes de persistance

"can't configure"

La configuration du serveur est incorrect (cf. message associé)

"can't restore messages"

La restauration du serveur est impossible (du soit à des problèmes de persistance, soit des problème de class)

20.3.2. Running

"Abnormal termination for <ThreadGroup>.<ThreadName>" + exception

Indicate that the Java Virtual Machine is broken or has run out of resources necessary for it to continue operating.
The server exits.

<ComponentName> + " Transaction problem" + exception

Permanent issues with transactional persistence component.

20.3.3. Stopping

"NTransactionLogFile, can't delete lockfile: "

nom du fichier donné

20.4. Error's messages client side

20.4.1. Administration

Démarrage ou configuration via un script XML - "fatal error parsing"

Identification du fichier, ligne et position de l'erreur données.

21. Deprecated List

Class org.objectweb.joram.client.jms.admin.AdminHelper

Member org.objectweb.joram.client.jms.admin.AdminModule.collocatedConnect (String name, String password, String identityClass)

Next to Joram 5.2 use connect methods with ConnectionFactory.

Member org.objectweb.joram.client.jms.admin.AdminModule.connect (String host, int port, String name, String password, int cnxTimer, String reliableClass, String identityClass)

Next to Joram 5.2 use connect methods with ConnectionFactory.

Member org.objectweb.joram.client.jms.admin.AdminModule.connect (javax.jms.TopicConnectionFactory cf, String name, String password)

No longer use TopicConnectionFactory next to Joram 5.2.

Member org.objectweb.joram.client.jms.admin.AdminModule.connect (javax.jms.TopicConnectionFactory cf, String name, String password, String identityClass)

No longer use TopicConnectionFactory next to Joram 5.2.

Member org.objectweb.joram.client.jms.admin.AdminModule.connect (String name, String password, int cnxTimer)

Next to Joram 5.2 use connect methods with ConnectionFactory.

Member org.objectweb.joram.client.jms.admin.AdminModule.connect (String host, int port, String name, String password, int cnxTimer)

Next to Joram 5.2 use connect methods with ConnectionFactory.

Member org.objectweb.joram.client.jms.admin.AdminModule.connect (String name, String password, int cnxTimer, String reliableClass)

Next to Joram 5.2 use connect methods with ConnectionFactory.

Member org.objectweb.joram.client.jms.admin.AdminModule.connect (String host, int port, String name, String password, int cnxTimer, String reliableClass)

Next to Joram 5.2 use connect methods with ConnectionFactory.

Member org.objectweb.joram.client.jms.admin.AdminModule.getDestinationsList ()

No longer supported next to Joram 5.2

Member org.objectweb.joram.client.jms.admin.AdminModule.getDestinationsList (int serverId)

No longer supported next to Joram 5.2

Member org.objectweb.joram.client.jms.admin.AdminModule.getServersIds ()

No longer supported next to Joram 5.2

Member org.objectweb.joram.client.jms.admin.AdminModule.getServersIds (String domain)

No longer supported next to Joram 5.2

Member org.objectweb.joram.client.jms.admin.AdminModule.getUsersList ()

No longer supported next to Joram 5.2

Member org.objectweb.joram.client.jms.admin.AdminModule.getUsersList (int serverId)

No longer supported next to Joram 5.2

Member `org.objectweb.joram.client.jms.admin.AdminWrapper.createDeadMQueue (int serverId, String name)`
No longer needed, any queue can be used as DMQ.

Class `org.objectweb.joram.client.jms.admin.DeadMQueue`
Since Joram 5.2.2 the DeadMQueue is a simply a Queue.

Member `org.objectweb.joram.client.jms.admin.User.readMessage (String subName, String msgId)`

Member `org.objectweb.joram.client.jms.Destination.getStatistic ()`

Class `org.objectweb.joram.client.jms.local.QueueLocalConnectionFactory`
Replaced next to Joram 5.2.1 by LocalConnectionFactory.

Class `org.objectweb.joram.client.jms.local.TopicLocalConnectionFactory`
Replaced next to Joram 5.2.1 by LocalConnectionFactory.

Class `org.objectweb.joram.client.jms.local.XALocalConnectionFactory`
Replaced next to Joram 5.2.1 by LocalConnectionFactory.

Class `org.objectweb.joram.client.jms.local.XAQueueLocalConnectionFactory`
Replaced next to Joram 5.2.1 by LocalConnectionFactory.

Class `org.objectweb.joram.client.jms.local.XATopicLocalConnectionFactory`
Replaced next to Joram 5.2.1 by LocalConnectionFactory.

Member `org.objectweb.joram.client.jms.Queue.readMessage (String msgId)`
Since Joram 5.2 use getMessage.

Member `org.objectweb.joram.client.jms.Queue.removeClusteredQueue (Queue removedQueue)`

Class `org.objectweb.joram.client.jms.QueueConnectionFactory`
Replaced next to Joram 5.2.1 by ConnectionFactory.

Class `org.objectweb.joram.client.jms.tcp.QueueTcpConnectionFactory`
Replaced next to Joram 5.2.1 by TcpConnectionFactory.

Class `org.objectweb.joram.client.jms.tcp.TopicTcpConnectionFactory`
Replaced next to Joram 5.2.1 by TcpConnectionFactory.

Class `org.objectweb.joram.client.jms.tcp.XAQueueTcpConnectionFactory`
Replaced next to Joram 5.2.1 by TcpConnectionFactory.

Class `org.objectweb.joram.client.jms.tcp.XATcpConnectionFactory`
Replaced next to Joram 5.2.1 by TcpConnectionFactory.

Class `org.objectweb.joram.client.jms.tcp.XATopicTcpConnectionFactory`
Replaced next to Joram 5.2.1 by TcpConnectionFactory.

Class `org.objectweb.joram.client.jms.TopicConnectionFactory`
Replaced next to Joram 5.2.1 by ConnectionFactory.

Class `org.objectweb.joram.client.jms.XAConnectionFactory`
Replaced next to Joram 5.2.1 by ConnectionFactory.

Class `org.objectweb.joram.client.jms.XAQueueConnectionFactory`
Replaced next to Joram 5.2.1 by ConnectionFactory.

Class `org.objectweb.joram.client.jms.XATopicConnectionFactory`
Replaced next to Joram 5.2.1 by ConnectionFactory.