Using Java
Third edition (January 2004)

This is the third edition of this book that applies to WebSphere MQ. It applies to the following products:

- IBM WebSphere MQ for AIX, Version 5.3
- IBM WebSphere MQ for HP-UX, Version 5.3
- IBM WebSphere MQ for iSeries, Version 5.3
- IBM WebSphere MQ for Linux for Intel, Version 5.3
- IBM WebSphere MQ for Linux for zSeries, Version 5.3
- IBM WebSphere MQ for Solaris, Version 5.3
- IBM WebSphere MQ for Windows, Version 5.3
- IBM WebSphere MQ for z/OS, Version 5.3

with fix pack 6 (CSD06) or later, and to any subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 1997, 2004. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
Contents

Figures ................................................................. ix

Tables ................................................................. xi

About this book ....................................................... xiii
Who this book is for .................................................... xiii
What you need to know to understand this book xiii
How to use this book .................................................... xiii
Terms used in this book ................................................ xiv

Summary of changes .................................................. xv
Changes for this edition (SC34-6066-02) .............................................................. xv
Changes for the second edition (SC34-6066-01) .................................................... xvi
Changes for the first edition (SC34-6066-00) .................................................... xvi

Part 1. Guidance for users ............................................ 1

Chapter 1. Getting started ........................................... 3
What are WebSphere MQ classes for Java? .................................................... 3
What are WebSphere MQ classes for Java Message Service? .................................... 3
Who should use WebSphere MQ Java? .................................................... 4
Connection options ..................................................... 4
  Client connection .................................................. 5
  Bindings connection ................................................ 5
Prerequisites ............................................................. 6

Chapter 2. Installation ................................................ 9
What is installed ....................................................... 9
  Installation directories .................................................. 10
  Environment variables ................................................ 10
  STEPLIB configuration on z/OS and OS/390 ................................................ 12
Web server configuration ................................................ 12
Running WebSphere MQ Java applications under the Java 2 Security Manager ............ 13

Chapter 3. Using WebSphere MQ classes for Java (WebSphere MQ base Java) ............. 15
Configuring your queue manager to accept client connections .................. 15
  TCP/IP client ....................................................... 15
Verifying with the sample application .................................................... 16
Running your own WebSphere MQ base Java programs .................................................... 17
Solving WebSphere MQ base Java problems .................................................... 17
  Tracing the sample application .................................................. 17
  Error messages ..................................................... 18

Chapter 4. Using WebSphere MQ classes for Java Message Service (WebSphere MQ JMS) .................................................... 19
JMS Postcard .......................................................... 19
  Setting up JMS Postcard .................................................. 19
  Starting ................................................................. 19
  Sign-on ................................................................. 20
  Sending a postcard .................................................... 20
  JMS Postcard configuration .................................................. 22
  How JMS Postcard works .................................................. 22
  Post installation setup .................................................. 25
  Additional setup for publish/subscribe mode .................................................. 26
  Queues that require authorization for non-privileged users .................. 29
Using the sample JMS applet to verify the TCP/IP client .................................................... 29
  Using the sample applet with OS/400 .................................................. 30
  Running the sample applet .................................................. 30
  Running the point-to-point IVT .................................................. 31
  Point-to-point verification without JNDI .................................................. 31
  Point-to-point verification with JNDI .................................................. 32
  IVT error recovery ..................................................... 34
The publish/subscribe installation verification test .................................................. 35
  Publish/subscribe verification without JNDI .................................................. 35
  Publish/subscribe verification with JNDI .................................................. 36
  PSIVT error recovery ..................................................... 37
Running your own WebSphere MQ JMS programs .................................................... 38
Solving problems .......................................................... 38
  Tracing programs ..................................................... 38
  Logging ................................................................. 39

Chapter 5. Using the WebSphere MQ JMS administration tool ........................................ 41
Invoking the administration tool .................................................... 41
Configuration .......................................................... 42
  Using an unlisted InitialContextFactory .................................................. 43
  Security ............................................................... 43
Configuring for WebSphere Application Server V3.5 .................................................. 44
Administration commands .................................................... 44
  Manipulating subcontexts .................................................. 45
  Administering JMS objects .................................................. 45
    Object types ......................................................... 45
    Verbs used with JMS objects .................................................. 47
    Creating objects ..................................................... 48
    Properties .......................................................... 49
    Property dependencies .................................................. 56
    The ENCODING property .................................................. 57
    SSL properties .......................................................... 58
    Sample error conditions .................................................. 59

Part 2. Programming with WebSphere MQ base Java ........................................... 61

Chapter 6. Introduction for programmers .................................................... 63
Why should I use the Java interface? .................................................... 63

© Copyright IBM Corp. 1997, 2004
Chapter 7. Writing WebSphere MQ base
Java programs ........................................ 67

Chapter 8. Environment-dependent
behavior ............................................. 95

Chapter 9. The WebSphere MQ base
Java classes and interfaces ........................ 101

The WebSphere MQ classes for Java interface ... 64
Java Development Kit ............................. 64
WebSphere MQ classes for Java class library ... 65

Should I write applets or applications? .............. 67
Connection differences .............................. 67
Client connections ..................................... 67
Bindings mode ....................................... 68
Defining which connection to use ..................... 68
Specifying a range of ports for client connections 68
Example code fragments ........................... 69
Example application code .......................... 72
Operations on queue managers ..................... 74
Setting up the WebSphere MQ environment ...... 74
Connecting to a queue manager .................... 75
Accessing queues and processes .................. 75
Handling messages .................................. 76
Handling errors ...................................... 77
Getting and setting attribute values ............... 78
Multithreaded programs ............................ 79
Writing user exits ................................... 79
Connection pooling ................................... 80
The default connection pool and multiple components ...... 83
Suppuling a different connection pool .......... 84
Supplying your own ConnectionManager .......... 85
JTA/JDBC coordination using WebSphere MQ base Java ................................. 87
Installation ......................................... 87
Usage ............................................... 88
Known problems and limitations ................. 88
Secure Sockets Layer (SSL) support .............. 89
Enabling SSL ........................................ 90
Using the distinguished name of the queue manager ...... 90
Using certificate revocation lists ................ 91
Supplying a customized SSLContext Factory ..... 92
Error handling when using SSL .................. 92
Compiling and testing WebSphere MQ base Java programs ......................... 93
Running WebSphere MQ base Java applets ........ 93
Running WebSphere MQ base Java applications 94
Tracing WebSphere MQ base Java programs .... 94

MQQueueManager.begin() method ............... 98
MQGetMessageOptions fields ..................... 98
Distribution lists ................................. 98
MQPutMessageOptions fields ..................... 98
MQMD fields ....................................... 99

MQChannelDefinition
Variables ........................................... 102
Constructors ....................................... 103
MQChannelExit
Variables ......................................... 104
Constructors ....................................... 106
MQDistributionList
Constructors ....................................... 107
Methods ........................................... 107
MQDistributionListItem
Variables .......................................... 109
Constructors ....................................... 109
MQEnvironment
Variables .......................................... 110
Constructors ....................................... 114
Methods ........................................... 114
MQException
Variables .......................................... 117
Constructors ....................................... 117
Methods ........................................... 118
MQGetMessageOptions
Variables .......................................... 119
Constructors ....................................... 122
Methods ........................................... 122
MQManagedObject
Variables .......................................... 123
Constructors ....................................... 124
Methods ........................................... 124
MQMessage
Variables .......................................... 126
Constructors ....................................... 134
Methods ........................................... 134
MQMessageTracker
Variables .......................................... 144
Constructors ....................................... 144
Methods ........................................... 146
MQPoolServices
Constructors ....................................... 146
Methods ........................................... 146
MQPoolServicesEvent
Variables .......................................... 147
Constructors ....................................... 147
Methods ........................................... 148
MQPoolToken
Constructors ....................................... 149
Methods ........................................... 149
MQProcess
Constructors ....................................... 150
Methods ........................................... 150
MQPutMessageOptions
Variables .......................................... 152
Constructors ....................................... 154
MQQueue
Constructors ....................................... 155
Methods ........................................... 155
MQQueueManager
Variables .......................................... 163

 MQChannelDefinition
Variables ........................................... 102
Constructors ....................................... 103
MQChannelExit
Variables ......................................... 104
Constructors ....................................... 106
MQDistributionList
Constructors ....................................... 107
Methods ........................................... 107
MQDistributionListItem
Variables .......................................... 109
Constructors ....................................... 109
MQEnvironment
Variables .......................................... 110
Constructors ....................................... 114
Methods ........................................... 114
MQException
Variables .......................................... 117
Constructors ....................................... 117
Methods ........................................... 118
MQGetMessageOptions
Variables .......................................... 119
Constructors ....................................... 122
Methods ........................................... 122
MQManagedObject
Variables .......................................... 123
Constructors ....................................... 124
Methods ........................................... 124
MQMessage
Variables .......................................... 126
Constructors ....................................... 134
Methods ........................................... 134
MQMessageTracker
Variables .......................................... 144
Constructors ....................................... 144
Methods ........................................... 146
MQPoolServices
Constructors ....................................... 146
Methods ........................................... 146
MQPoolServicesEvent
Variables .......................................... 147
Constructors ....................................... 147
Methods ........................................... 148
MQPoolToken
Constructors ....................................... 149
Methods ........................................... 149
MQProcess
Constructors ....................................... 150
Methods ........................................... 150
MQPutMessageOptions
Variables .......................................... 152
Constructors ....................................... 154
MQQueue
Constructors ....................................... 155
Methods ........................................... 155
MQQueueManager
Variables .......................................... 163

MQChannelDefinition
Variables ........................................... 102
Constructors ....................................... 103
MQChannelExit
Variables ......................................... 104
Constructors ....................................... 106
MQDistributionList
Constructors ....................................... 107
Methods ........................................... 107
MQDistributionListItem
Variables .......................................... 109
Constructors ....................................... 109
MQEnvironment
Variables .......................................... 110
Constructors ....................................... 114
Methods ........................................... 114
MQException
Variables .......................................... 117
Constructors ....................................... 117
Methods ........................................... 118
MQGetMessageOptions
Variables .......................................... 119
Constructors ....................................... 122
Methods ........................................... 122
MQManagedObject
Variables .......................................... 123
Constructors ....................................... 124
Methods ........................................... 124
MQMessage
Variables .......................................... 126
Constructors ....................................... 134
Methods ........................................... 134
MQMessageTracker
Variables .......................................... 144
Constructors ....................................... 144
Methods ........................................... 146
MQPoolServices
Constructors ....................................... 146
Methods ........................................... 146
MQPoolServicesEvent
Variables .......................................... 147
Constructors ....................................... 147
Methods ........................................... 148
MQPoolToken
Constructors ....................................... 149
Methods ........................................... 149
MQProcess
Constructors ....................................... 150
Methods ........................................... 150
MQPutMessageOptions
Variables .......................................... 152
Constructors ....................................... 154
MQQueue
Constructors ....................................... 155
Methods ........................................... 155
MQQueueManager
Variables .......................................... 163

MQChannelDefinition
Variables ........................................... 102
Constructors ....................................... 103
MQChannelExit
Variables ......................................... 104
Constructors ....................................... 106
MQDistributionList
Constructors ....................................... 107
Methods ........................................... 107
MQDistributionListItem
Variables .......................................... 109
Constructors ....................................... 109
MQEnvironment
Variables .......................................... 110
Constructors ....................................... 114
Methods ........................................... 114
MQException
Variables .......................................... 117
Constructors ....................................... 117
Methods ........................................... 118
MQGetMessageOptions
Variables .......................................... 119
Constructors ....................................... 122
Methods ........................................... 122
MQManagedObject
Variables .......................................... 123
Constructors ....................................... 124
Methods ........................................... 124
MQMessage
Variables .......................................... 126
Constructors ....................................... 134
Methods ........................................... 134
MQMessageTracker
Variables .......................................... 144
Constructors ....................................... 144
Methods ........................................... 146
MQPoolServices
Constructors ....................................... 146
Methods ........................................... 146
MQPoolServicesEvent
Variables .......................................... 147
Constructors ....................................... 147
Methods ........................................... 148
MQPoolToken
Constructors ....................................... 149
Methods ........................................... 149
MQProcess
Constructors ....................................... 150
Methods ........................................... 150
MQPutMessageOptions
Variables .......................................... 152
Constructors ....................................... 154
MQQueue
Constructors ....................................... 155
Methods ........................................... 155
MQQueueManager
Variables .......................................... 163
Chapter 13. JMS messages . . . . . . . 257
Message selectors . . . . . . . 257
Mapping JMS messages onto WebSphere MQ messages . . . . . . . 261
  The MQRFH2 header . . . . . . . 262
  JMS fields and properties with corresponding MQMD fields . . . . . . . 265
Mapping JMS fields onto WebSphere MQ fields (outgoing messages) . . . . . . . 266
Mapping WebSphere MQ fields onto JMS fields (incoming messages) . . . . . . . 271
Mapping JMS to a native WebSphere MQ application . . . . . . . 273
Message body . . . . . . . 273

Chapter 14. WebSphere MQ JMS Application Server Facilities . . . . . . . 277
ASF classes and functions . . . . . . . 277
  ConnectionConsumer . . . . . . . 277
  Planning an application . . . . . . . 278
  Error handling . . . . . . . 282
Application server sample code . . . . . . . 283
  MyServerSession.java . . . . . . . 285
  MyServerSessionPool.java . . . . . . . 285
  MessageListenerFactory.java . . . . . . . 286
Examples of ASF use . . . . . . . 287
  Load1.java . . . . . . . 287
  CountingMessageListenerFactory.java . . . . . . . 288
  ASFClient1.java . . . . . . . 289
  Load2.java . . . . . . . 290
  LoggingMessageListenerFactory.java . . . . . . . 290
  ASFClient2.java . . . . . . . 290
  TopicLoad.java . . . . . . . 291
  ASFClient3.java . . . . . . . 292
  ASFClient4.java . . . . . . . 293
  ASFClient5.java . . . . . . . 294

Chapter 15. JMS interfaces and classes . . . . . . . . . . . . . 295
Sun Java Message Service classes and interfaces . . . . . . . 295
WebSphere MQ JMS classes . . . . . . . 298
BytesMessage . . . . . . . 300
  Methods . . . . . . . 300
  Cleanup . . . . . . . 308
    WebSphere MQ constructor . . . . . . . 308
    Methods . . . . . . . 308
Connection . . . . . . . 313
  Methods . . . . . . . 313
ConnectionConsumer . . . . . . . 318
  Methods . . . . . . . 318
ConnectionFactory . . . . . . . 319
  WebSphere MQ constructor . . . . . . . 319
  Methods . . . . . . . 319
ConnectionMetaData . . . . . . . 335
  WebSphere MQ constructor . . . . . . . 335
  Methods . . . . . . . 335
DeliveryMode . . . . . . . 337
  Fields . . . . . . . 337
Destination . . . . . . . 338
  WebSphere MQ constructors . . . . . . . 338
Methods . . . . . . . 338
ExceptionListener . . . . . . . 340
  Methods . . . . . . . 340
MapMessage . . . . . . . 341
  Methods . . . . . . . 341
Message . . . . . . . 349
  Fields . . . . . . . 349
  Methods . . . . . . . 349
MessageConsumer . . . . . . . 363
  Methods . . . . . . . 363
MessageListener . . . . . . . 366
  Methods . . . . . . . 366
MessageProducer . . . . . . . 367
  WebSphere MQ constructors . . . . . . . 367
  Methods . . . . . . . 367
MQQueueEnumeration . . . . . . . 373
  Methods . . . . . . . 373
ObjectName . . . . . . . 374
  Methods . . . . . . . 374
Queue . . . . . . . 375
  WebSphere MQ constructors . . . . . . . 375
  Methods . . . . . . . 375
QueueBrowser . . . . . . . 377
  Methods . . . . . . . 377
QueueConnection . . . . . . . 379
  Methods . . . . . . . 379
QueueConnectionFactory . . . . . . . 381
  WebSphere MQ constructor . . . . . . . 381
  Methods . . . . . . . 381
QueueReceiver . . . . . . . 384
  Methods . . . . . . . 384
QueueRequestor . . . . . . . 385
  Constructors . . . . . . . 385
  Methods . . . . . . . 385
QueueSender . . . . . . . 387
  Methods . . . . . . . 387
QueueSession . . . . . . . 390
  Methods . . . . . . . 390
Session . . . . . . . 393
  Fields . . . . . . . 393
  Methods . . . . . . . 393
StreamMessage . . . . . . . 405
  Methods . . . . . . . 405
TemporaryQueue . . . . . . . 413
  Methods . . . . . . . 413
TemporaryTopic . . . . . . . 414
  WebSphere MQ constructor . . . . . . . 414
  Methods . . . . . . . 414
TextMessage . . . . . . . 415
  Methods . . . . . . . 415
Topic . . . . . . . 416
  WebSphere MQ constructor . . . . . . . 416
  Methods . . . . . . . 416
TopicConnectionFactory . . . . . . . 420
  WebSphere MQ constructor . . . . . . . 420
  Methods . . . . . . . 420
TopicPublisher . . . . . . . 431
  Methods . . . . . . . 431
TopicRequestor . . . . . . . 434
Figures

1. WebSphere MQ classes for Java example applet ........................................ 70
2. WebSphere MQ classes for Java example application .................................. 73
3. WebSphere MQ classes for Java Message Service topic name hierarchy .......... 221
4. How messages are transformed between JMS and WebSphere MQ using the MQRFH2 header ............................................................. 261
5. How JMS messages are transformed to WebSphere MQ messages (no MQRFH2 header) ................................................................. 273
6. ServerSessionPool and ServerSession functionality ....................................... 284
7. WebSphere MQ Integrator message flow ....................................................... 470
Tables

1. Platforms and connection modes .................................. 5
2. Product installation directories ................................... 10
3. Samples directories ..................................................... 10
4. Sample CLASSPATH statements for the product .................. 11
5. Environment variables for the product ............................. 12
6. Classes that are tested by IVT ...................................... 34
7. Administration verbs .................................................. 44
8. Syntax and description of commands used to manipulate subcontexts ........................................ 45
9. The JMS object types that are handled by the administration tool ........................................... 46
10. Syntax and description of commands used to manipulate administered objects .......................... 47
11. Property names and valid values .................................... 49
12. The valid combinations of property and object type ................. 53
13. Character set identifiers ............................................. 127
14. Set methods on MQQueueConnectionFactory .................... 202
15. Property names for queue and topic URIs ......................... 205
16. Symbolic values for queue properties ............................... 206
17. The JMS 1.1 domain independent interfaces ....................... 235
18. Possible values for NameValueCCSID field ....................... 263
19. MQRFH2 folders and properties used by JMS ..................... 263
20. Property datatype values and definitions .......................... 264
21. JMS header fields mapping to MQMD fields ....................... 265
22. JMS properties mapping to MQMD fields .......................... 266
23. JMS provider specific properties mapping to MQMD fields ....... 266
24. Outgoing message field mapping .................................... 267
25. Outgoing message JMS property mapping ........................ 267
26. Outgoing message JMS provider specific property mapping ...... 267

27. Incoming message JMS header field mapping .................... 272
28. Incoming message property mapping ............................... 272
29. Incoming message provider specific JMS property mapping .... 272
30. Load1 parameters and defaults ...................................... 288
31. ASFClient1 parameters and defaults ............................... 289
32. TopicLoad parameters and defaults ................................. 291
33. ASFClient3 parameters and defaults ............................... 292
34. Summary of interfaces in package javax.jms ...................... 295
35. Summary of classes in package javax.jms ......................... 297
36. Summary of classes in package com.ibm.mq.jms .................. 298
37. Summary of classes in package com.ibm.jms ...................... 299
38. Comparison of representations of property values within the administration tool and within programs .... 457
39. Utilities supplied with WebSphere MQ classes for Java Message Service ................ 461
40. Attribute settings for javaCodebase ............................... 464
41. Attribute settings for javaClassName ............................. 464
42. Attribute settings for javaClassNames ............................ 464
43. Attribute settings for javaFactory ................................. 465
44. Attribute settings for javaReferenceAddress ..................... 465
45. Attribute settings for javaSerializedData ....................... 465
46. objectClass definition for javaSerializedObject ............... 465
47. objectClass definition for javaObject ............................ 466
48. objectClass definition for javaContainer ........................ 466
49. objectClass definition for javaNamingReference ................. 466
50. CipherSpecs and matching CipherSuites .......................... 487
About this book

This book describes:

- WebSphere® MQ classes for Java™, which can be used to access WebSphere MQ systems
- WebSphere MQ classes for Java Message Service, which can be used to access both Java Message Service (JMS) and WebSphere MQ applications

Note: Consult the README file for information that expands and corrects information in this book. The README file is installed with the WebSphere MQ Java code and can be found in the doc subdirectory.

Who this book is for

This information is written for programmers who are familiar with the procedural WebSphere MQ application programming interface as described in the WebSphere MQ Application Programming Guide. It shows how to transfer this knowledge to become productive with the WebSphere MQ Java programming interfaces.

What you need to know to understand this book

You need:

- Knowledge of the Java programming language
- Understanding of the purpose of the message queue interface (MQI) as described in the WebSphere MQ Application Programming Guide and the chapter about Call Descriptions in the WebSphere MQ Application Programming Reference
- Experience of WebSphere MQ programs in general, or familiarity with the content of the other WebSphere MQ publications

Users intending to use the WebSphere MQ base Java with CICS® Transaction Server for OS/390® also need to be familiar with:

- Customer Information Control System (CICS) concepts
- Using the CICS Java Application Programming Interface (API)
- Running Java programs from within CICS

Users intending to use VisualAge® for Java to develop OS/390 UNIX® System Services High Performance Java (HPJ) applications should be familiar with the Enterprise Toolkit for OS/390 (supplied with VisualAge for Java Enterprise Edition for OS/390, Version 2).

How to use this book

Part 1 of this book tells you how to use WebSphere MQ base Java and WebSphere MQ JMS; Part 2 helps programmers wanting to use WebSphere MQ base Java; Part 3 helps programmers wanting to use WebSphere MQ JMS.

First, read the chapters in Part 1 that introduce you to WebSphere MQ base Java and WebSphere MQ JMS. Then use the programming guidance in Part 2 or 3 to understand how to use the classes to send and receive WebSphere MQ messages in the environment you want to use.
How to use this book

Remember to check the README file installed with the WebSphere MQ Java code for later or more specific information for your environment.

Terms used in this book

The term WebSphere MQ base Java means WebSphere MQ classes for Java.

The term WebSphere MQ JMS means WebSphere MQ classes for Java Message Service.

The term WebSphere MQ Java means WebSphere MQ classes for Java and WebSphere MQ classes for Java Message Service combined.

The term Version 5.3 products means:
• WebSphere MQ for AIX®, Version 5.3
• WebSphere MQ for HP-UX, Version 5.3
• WebSphere MQ for iSeries™, Version 5.3
• WebSphere MQ for Linux for Intel™, Version 5.3
• WebSphere MQ for Linux for zSeries™, Version 5.3
• WebSphere MQ for Sun Solaris, Version 5.3
• WebSphere MQ for Windows®, Version 5.3
• WebSphere MQ for z/OS®, Version 5.3

The term WebSphere MQ for UNIX systems means:
• WebSphere MQ for AIX
• WebSphere MQ for HP-UX
• WebSphere MQ for Linux for Intel
• WebSphere MQ for Linux for zSeries
• WebSphere MQ for Sun Solaris

UNIX systems is also used as a general term for the UNIX platforms.

The term WebSphere MQ for Windows systems means WebSphere MQ running on the following Windows platforms:
• Windows NT®
• Windows 2000
• Windows XP

Windows systems, or just Windows, is also used as a general term for these Windows platforms.
Summary of changes

This section describes changes in this edition of WebSphere MQ Using Java. Changes since the previous edition of the book are marked by vertical lines to the left of the changes.

<table>
<thead>
<tr>
<th>Changes for this edition (SC34–6066–02)</th>
</tr>
</thead>
<tbody>
<tr>
<td>This edition includes documentation to support the following new function:</td>
</tr>
<tr>
<td>• An implementation of Version 1.1 of the JMS API specification</td>
</tr>
<tr>
<td>• Direct connection to a WebSphere Business Integration Event Broker or</td>
</tr>
<tr>
<td>WebSphere Business Integration Message Broker broker using:</td>
</tr>
<tr>
<td>– SSL authentication</td>
</tr>
<tr>
<td>– Multicast</td>
</tr>
<tr>
<td>– HTTP tunnelling</td>
</tr>
<tr>
<td>– Connect via proxy</td>
</tr>
<tr>
<td>• Connecting to a WebSphere MQ queue manager through a firewall</td>
</tr>
<tr>
<td>• Sparse subscriptions</td>
</tr>
<tr>
<td>• Message selection by the broker</td>
</tr>
<tr>
<td>• User defined prefixes for WebSphere MQ dynamic queues</td>
</tr>
<tr>
<td>• Configuring a connection pool for an MQSimpleConnectionManager object in</td>
</tr>
<tr>
<td>WebSphere MQ classes for Java</td>
</tr>
</tbody>
</table>

This edition also contains various editorial improvements, clarifications, and corrections.

<table>
<thead>
<tr>
<th>Changes for the second edition (SC34-6066-01)</th>
</tr>
</thead>
<tbody>
<tr>
<td>This edition includes the following changes:</td>
</tr>
<tr>
<td>• A section listing JMS exception messages.</td>
</tr>
<tr>
<td>See Appendix I, “JMS exception messages,” on page 489.</td>
</tr>
<tr>
<td>• Miscellaneous corrections and clarifications.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Changes for the first edition (SC34-6066-00)</th>
</tr>
</thead>
<tbody>
<tr>
<td>This edition includes the following changes:</td>
</tr>
<tr>
<td>• Changes for</td>
</tr>
<tr>
<td>– Integration with the WebSphere MQ product</td>
</tr>
<tr>
<td>– JMS Postcard</td>
</tr>
<tr>
<td>– Secure Sockets Layer (SSL) support</td>
</tr>
<tr>
<td>• Miscellaneous corrections and clarifications.</td>
</tr>
</tbody>
</table>
Changes
Part 1. Guidance for users

Chapter 1. Getting started ........................................... 3
What are WebSphere MQ classes for Java? ........................ 3
What are WebSphere MQ classes for Java Message Service? .... 3
Who should use WebSphere MQ Java? ............................ 4
Connection options ............................................... 4
  Client connection ........................................... 5
  Bindings connection ....................................... 5
Prerequisites .................................................... 6

Chapter 2. Installation ............................................... 9
What is installed .................................................. 9
  Installation directories .................................... 10
  Environment variables .................................... 10
  STEPLIB configuration on z/OS and OS/390 .................. 12
Web server configuration ....................................... 12
Running WebSphere MQ Java applications under the Java 2 Security Manager .................................. 13

Chapter 3. Using WebSphere MQ classes for Java (WebSphere MQ base Java) .................................. 15
Configuring your queue manager to accept client connections ......................................................... 15
  TCP/IP client .............................................. 15
Verifying with the sample application ......................... 16
Running your own WebSphere MQ base Java programs .......................................................... 17
Solving WebSphere MQ base Java problems .................. 17
  Tracing the sample application ............................ 17
  Error messages ........................................... 18

Chapter 4. Using WebSphere MQ classes for Java Message Service (WebSphere MQ JMS) ................... 19
JMS Postcard .................................................... 19
  Setting up JMS Postcard ................................... 19
  Starting .................................................... 19
  Sign-on .................................................... 20
    Sign-on advanced options ................................ 20
  Sending a postcard ....................................... 20
    Running JMS Postcard with one queue manager ....... 20
    Running JMS Postcard with two queue managers ...... 21
JMS Postcard configuration .................................... 22
  JMS Postcard default configuration ....................... 22
How JMS Postcard works ........................................ 22
  Starting .................................................... 22
  Receiving messages ....................................... 23
  Sending messages ....................................... 23
  How the postcards get there ................................ 23
  Tidying up undeliverable messages ....................... 24
  Exchanging messages between different WebSphere MQ Postcard applications ............................... 24
  Customizing JMS Postcard ................................ 24
Post installation setup ........................................... 25
  Additional setup for publish/subscribe mode ............... 26
    For a broker running on a remote queue manager ...... 28
  Queues that require authorization for non-privileged users ..................................................... 29
Using the sample JMS applet to verify the TCP/IP client ........ 29
  Using the sample applet with OS/400 ..................... 30
  Running the sample applet ................................ 30
  Tracing the sample as an application .................... 31
Running the point-to-point IVT ................................ 31
  Point-to-point verification without JNDI .................. 31
  Point-to-point verification with JNDI .................... 32
VF error recovery ........................................... 34
The publish/subscribe installation verification test ........ 35
  Publish/subscribe verification without JNDI ............ 35
  Publish/subscribe verification with JNDI ................ 36
PSIVT error recovery ......................................... 37
Running your own WebSphere MQ JMS programs ............... 38
Solving problems ............................................. 38
  Tracing programs ....................................... 38
  Logging .................................................. 39

Chapter 5. Using the WebSphere MQ JMS administration tool ..................................................... 41
Invoking the administration tool ................................ 41
Configuration .................................................. 42
  Using an unlisted InitialContextFactory .................. 43
  Security .................................................. 43
Configuring for WebSphere Application Server V3.5 ........ 44
Administration commands ..................................... 44
Manipulating subcontexts ..................................... 45
Administering JMS objects .................................... 45
  Object types ........................................... 45
  Verbs used with JMS objects .............................. 47
Creating objects ............................................. 48
  LDAP naming considerations .............................. 48
Properties ................................................... 49
Property dependencies ....................................... 56
The ENCODING property ..................................... 57
SSL properties .............................................. 58
Sample error conditions ...................................... 59
Chapter 1. Getting started

This chapter gives an overview of WebSphere MQ classes for Java and WebSphere MQ classes for Java Message Service and their uses.

What are WebSphere MQ classes for Java?

WebSphere MQ classes for Java (also referred to as WebSphere MQ base Java) allow a program written in the Java programming language to:

- Connect to WebSphere MQ as a WebSphere MQ client
- Connect directly to a WebSphere MQ server

WebSphere MQ base Java enables Java applets, applications, and servlets to issue calls and queries to WebSphere MQ. This gives access to mainframe and legacy applications, typically over the Internet, without necessarily having any other WebSphere MQ code on the client machine. With WebSphere MQ base Java, Internet users can become true participants in transactions, rather than just givers and receivers of information.

What are WebSphere MQ classes for Java Message Service?

WebSphere MQ classes for Java Message Service (also referred to as WebSphere MQ JMS) is a set of Java classes that implement Sun’s Java Message Service (JMS) interfaces to enable JMS programs to access WebSphere MQ systems. This book describes an implementation of Version 1.1 of the JMS API specification, which is backwards compatible with Version 1.0.2b. Any features of the implementation that apply only to Version 1.1 of the specification, and not to Version 1.0.2b, are clearly marked. Both the point-to-point and publish/subscribe models of JMS are supported.

Using WebSphere MQ JMS as the API to write WebSphere MQ applications has a number of benefits. Some advantages derive from JMS being an open standard with multiple implementations. Other advantages come from additional features that are present in WebSphere MQ JMS, but not in WebSphere MQ base Java.

Benefits arising from the use of an open standard include:
- The protection of investment, both in skills and application code
- The availability of people skilled in JMS application programming
- The ability to plug in different JMS implementations to fit different requirements

Sun’s Web site at http://java.sun.com provides more information about the benefits of the JMS API.

The extra function provided over WebSphere MQ base Java includes:
- Asynchronous message delivery. Messages can be delivered to an application as they arrive, on a separate thread.
- Message selectors.
- Support for publish/subscribe messaging.
- Structured, more abstract, message classes. Implementation details are left to the JMS provider.
Who should use WebSphere MQ Java?

If your enterprise fits any of the following scenarios, you can gain significant advantage by using WebSphere MQ classes for Java and WebSphere MQ classes for Java Message Service:

- A medium or large enterprise that is introducing intranet-based client/server solutions. Here, Internet technology provides low cost easy access to global communications; WebSphere MQ connectivity provides high integrity with assured delivery and time independence.

- A medium or large enterprise with a need for reliable business-to-business communications with partner enterprises. Here again, the Internet provides low-cost easy access to global communications; WebSphere MQ connectivity provides high integrity with assured delivery and time independence.

- A medium or large enterprise that wants to provide access from the public Internet to some of its enterprise applications. Here, the Internet provides global reach at a low cost; WebSphere MQ connectivity provides high integrity through the queuing paradigm. In addition to low cost, the business can achieve improved customer satisfaction through 24 hour a day availability, fast response, and improved accuracy.

- An Internet Service provider, or other Value Added Network provider. These companies can exploit the low cost and easy communications provided by the Internet. They can also add value with the high integrity provided by WebSphere MQ connectivity. An Internet Service provider that exploits WebSphere MQ can immediately acknowledge receipt of input data from a Web browser, guarantee delivery, and provide an easy way for the user of the Web browser to monitor the status of the message.

WebSphere MQ and WebSphere MQ classes for Java Message Service provide an excellent infrastructure to access enterprise applications and develop complex Web applications. A service request from a Web browser can be queued then processed when possible, allowing a timely response to be sent to the end user, regardless of system loading. By placing this queue close to the user in network terms, the load on the network does not impact the timeliness of the response. Also, the transactional nature of WebSphere MQ messaging means that a simple request from the browser can be expanded safely into a sequence of individual back end processes in a transactional manner.

WebSphere MQ classes for Java also enables application developers to exploit the power of the Java programming language to create applets and applications that can run on any platform that supports the Java runtime environment. These factors combine to reduce the development time for multi-platform WebSphere MQ applications significantly. Also, if there are enhancements to applets in the future, end users automatically pick these up as the applet code is downloaded.

Connection options

Programmable options allow WebSphere MQ Java to connect to WebSphere MQ in either of the following ways:

- As a WebSphere MQ client using Transmission Control Protocol/Internet Protocol (TCP/IP)
- In bindings mode, connecting directly to WebSphere MQ

Table 1 on page 5 shows which of these connection modes can be used for each platform.
In addition, WebSphere MQ JMS publish/subscribe applications can connect directly across TCP/IP to the IBM® WebSphere MQ Event Broker program. For more information about this connection see Chapter 11, “Writing WebSphere MQ JMS publish/subscribe applications,” on page 213.

Table 1. Platforms and connection modes

<table>
<thead>
<tr>
<th>Server platform</th>
<th>Client</th>
<th>Bindings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows NT</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Windows 2000</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Windows XP</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>AIX</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Solaris (v2.6, v2.8, V7, or SunOS v5.6, v5.7)</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>OS/400®</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>HP-UX</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>OS/390 and z/OS</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Linux on Intel</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Linux on zSeries</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

Notes:
1. HP-UX Java bindings support is available only for HP-UXv11 systems running the POSIX draft 10 pthreaded version of WebSphere MQ.
2. On Linux on zSeries, only TCP/IP client connectivity is supported.

The following sections describe these options in more detail.

Client connection

To use WebSphere MQ Java as a WebSphere MQ client, you can install it either on the WebSphere MQ server machine, which may also contain a Web server, or on a separate machine. If you install WebSphere MQ Java on the same machine as a Web server, you can download and run WebSphere MQ client applications on machines that do not have WebSphere MQ Java installed locally.

Wherever you choose to install the client, you can run it in three different modes:

From within any Java-enabled Web browser
In this mode, the locations of the WebSphere MQ queue managers that can be accessed are constrained by the security restrictions of the browser that is used.

Using an appletviewer
To use this method, you must have the Java Development Kit (JDK™) or Java Runtime Environment (JRE) installed on the client machine.

As a standalone Java program or in a Web application server
To use this method, you must have the Java Development Kit (JDK) or Java Runtime Environment (JRE) installed on the client machine.

Bindings connection

When used in bindings mode, WebSphere MQ Java uses the Java Native Interface (JNI) to call directly into the existing queue manager API, rather than communicating through a network. This provides better performance for
Connections

WebSphere MQ applications than using network connections. Unlike the client mode, applications that are written using the bindings mode cannot be downloaded as applets.

To use the bindings connection, you must install WebSphere MQ Java on the WebSphere MQ server.

Prerequisites

To run WebSphere MQ base Java, you need the following software:

- WebSphere MQ for the server platform you want to use.
- Java Development Kit (JDK) for the server platform.
- Java Development Kit, Java Runtime Environment (JRE), or Java-enabled Web browser for client platforms. (See “Client connection” on page 5.)
- For z/OS and OS/390, OS/390 Version 2 Release 9 or higher, or z/OS, with UNIX System Services (USS).
- For OS/400, the iSeries Developer Kit for Java, 5769-JV1, and the Qshell Interpreter, OS/400 (5769-SSI) Option 30.

The following list shows the supported Java 2 Software Development Kits and Java Runtime Environments:

- IBM Developer Kit for AIX, Java Technology Edition, Version 1.3.1
- IBM Developer Kit for Linux, Java Technology Edition, Version 1.3.1
- IBM Developer Kit for OS/390, Java Technology Edition, Version 1.3.1
- IBM iSeries Developer Kit for Java, Version 1.3
- HP-UX SDK, for the Java platform, Version 1.3.1
- Java 2 Standard Edition, for the Solaris Operating Environment, SDK 1.3.1

To fully support Secure Socket Layer (SSL) authentication, you need a Java Runtime Environment at Version 1.4.0 for your platform. SSL support enables WebSphere MQ Java and Java Message Service (JMS) applications to benefit from secure connection to the queue manager, providing authentication, message integrity, and data encryption.

Check the README file for the latest information about operating system levels this product has been tested against.

To use the WebSphere MQ JMS administration tool (see Chapter 5, “Using the WebSphere MQ JMS administration tool,” on page 41), you need one of the following service provider packages, supplied with WebSphere MQ:

- File system - fscontext.jar, providerutil.jar.

These packages provide the Java Naming and Directory Service (JNDI) service. This is the resource that stores physical representations of the administered objects. Users of WebSphere MQ JMS probably use an LDAP server for this purpose, but the tool also supports the use of the file system context service provider. If you use an LDAP server, configure it to store JMS objects. For information to assist with this configuration, refer to Appendix C, “LDAP schema definition for storing Java objects,” on page 463.

To use publish/subscribe applications, you need one of the following:

- SupportPac™ MA0C: MQSeries® Publish/Subscribe. You can find this at:
Prerequisites

www.ibm.com/software/ts/mqseries/txppacs/ma0c.html

- WebSphere MQ Integrator Version 2
- WebSphere MQ Event Broker Version 2.1
- WebSphere Business Integration Message Broker Version 5.0
- WebSphere Business Integration Event Broker Version 5.0

To use the XOpen/XA facilities of WebSphere MQ JMS on OS/400 you need a specific PTF. Check the README file for further information.
Chapter 2. Installation

This chapter tells you how to install the WebSphere MQ classes for Java and WebSphere MQ classes for Java Message Service code.

What is installed

The latest versions of both WebSphere MQ base Java and WebSphere MQ JMS (together known as WebSphere MQ Java) are installed with WebSphere MQ. You might need to override default installation options to make sure this is done.

Refer to the following books for more information about installing WebSphere MQ:

- WebSphere MQ for AIX, V5.3 Quick Beginnings
- WebSphere MQ for HP-UX, V5.3 Quick Beginnings
- WebSphere MQ for iSeries V5.3 Quick Beginnings
- WebSphere MQ for Linux, V5.3 Quick Beginnings
- WebSphere MQ for Sun Solaris, Version 5.3 Quick Beginnings
- WebSphere MQ for Windows NT and Windows 2000, Version 5.3 Quick Beginnings
- WebSphere MQ for z/OS Program Directory

WebSphere MQ base Java is contained in the following Java .jar files:

- com.ibm.mq.jar
  - This code includes support for all the connection options.

- com.ibm.mqbind.jar
  - This code supports only the bindings connection and is not supplied or supported on all platforms. We recommend that you do not use it in any new applications.

WebSphere MQ JMS is contained in the following Java .jar file:

- com.ibm.mqjms.jar

The following Java libraries from Sun Microsystems are distributed with the WebSphere MQ JMS product:

- connector.jar
  - Version 1.0

- fscontext.jar
  - Version 1.2

- jms.jar
  - Version 1.1

- jndi.jar
  - Version 1.2.1 (except for z/OS and OS/390)

- ldap.jar
  - Version 1.2.2 (except for z/OS and OS/390)

- providerutil.jar
  - Version 1.2

- jta.jar
  - Version 1.0.1

When installation is complete, files and samples are installed in the locations shown in "Installation directories” on page 10.

We also supply postcard.jar for the Postcard application; see “JMS Postcard” on page 19.
After installation, update your environment variables, as shown in "Environment variables."

Note: Do not install the product, then subsequently install or reinstall a version of SupportPac MA88, or your WebSphere MQ Java support might revert to an earlier level.

Installation directories

The WebSphere MQ Java V5.3 files are installed in the directories shown in Table 2.

Table 2. Product installation directories

<table>
<thead>
<tr>
<th>Platform</th>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>/usr/mqm/java/</td>
</tr>
<tr>
<td>z/OS and OS/390</td>
<td>install_dir/mqm/java/</td>
</tr>
<tr>
<td>iSeries and AS/400®</td>
<td>/QIBM/ProdData/mqm/java/</td>
</tr>
<tr>
<td>HP-UX and Solaris</td>
<td>/opt/mqm/java/</td>
</tr>
<tr>
<td>Linux</td>
<td>/opt/mqm/java/</td>
</tr>
<tr>
<td>Windows systems</td>
<td>\Program Files\IBM\WebSphere MQ\java</td>
</tr>
</tbody>
</table>

Note: On z/OS and OS/390, install_dir is the directory in which you installed the product; this is likely to be /usr/lpp.

Some sample programs, such as the Installation Verification Programs (IVP), are supplied. Table 3 lists the directory path to these on different platforms. WebSphere MQ base Java samples are within a subdirectory base and WebSphere MQ JMS samples are within a subdirectory jms.

Table 3. Samples directories

<table>
<thead>
<tr>
<th>Platform</th>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>/usr/mqm/samp/java/</td>
</tr>
<tr>
<td>z/OS and OS/390</td>
<td>install_dir/mqm/java/samples/</td>
</tr>
<tr>
<td>iSeries and AS/400®</td>
<td>/QIBM/ProdData/mqm/java/samples/</td>
</tr>
<tr>
<td>HP-UX and Solaris</td>
<td>/opt/mqm/samp/java/</td>
</tr>
<tr>
<td>Linux</td>
<td>/opt/mqm/samp/java/</td>
</tr>
<tr>
<td>Windows systems</td>
<td>\Program Files\IBM\WebSphere MQ\tools\Java</td>
</tr>
</tbody>
</table>

Note: On z/OS and OS/390, install_dir is the directory in which you installed the product; this is likely to be /usr/lpp.

Environment variables

After installation, update your CLASSPATH environment variable to include the WebSphere MQ base Java code and samples directories. Table 4 on page 11 shows typical CLASSPATH settings for the various platforms.

WebSphere MQ Java uses other environment variables. Some are platform dependent and are listed in Table 5 on page 12. MQ_JAVA_INSTALL_PATH and MQ_JAVA_DATA_PATH are common across platforms. On Windows systems, these variables are automatically set by the installation program, but on other platforms you need to set them manually to complete installation.
MQ_JAVA_INSTALL_PATH points to the product installation directory, as shown in Table 2 on page 10. MQ_JAVA_DATA_PATH points to the root directory for logging and tracing, and is included so that you can use the same directory for WebSphere MQ Java and the base WebSphere MQ product.

Table 4. Sample CLASSPATH statements for the product

<table>
<thead>
<tr>
<th>Platform</th>
<th>Sample CLASSPATH</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>CLASSPATH=/usr/mqm/java/lib/com.ibm.mq.jar:</td>
</tr>
<tr>
<td></td>
<td>/usr/mqm/java/lib/connector.jar:</td>
</tr>
<tr>
<td></td>
<td>/usr/mqm/samp/java/base:</td>
</tr>
<tr>
<td>HP-UX and Solaris</td>
<td>CLASSPATH=/opt/mqm/java/lib/com.ibm.mq.jar:</td>
</tr>
<tr>
<td></td>
<td>/opt/mqm/java/lib/connector.jar:</td>
</tr>
<tr>
<td></td>
<td>/opt/mqm/samp/java/base:</td>
</tr>
<tr>
<td>Windows systems</td>
<td>CLASSPATH=mg_root_dir[4]java\lib\com.ibm.mq.jar;</td>
</tr>
<tr>
<td></td>
<td>mg_root_dir[4]java\lib\connector.jar;</td>
</tr>
<tr>
<td></td>
<td>mg_root_dir[4]tools\java\base\java\lib\jta.jar;</td>
</tr>
<tr>
<td>z/OS and OS/390</td>
<td>CLASSPATH=install_dir[3]mqm/java/lib/com.ibm.mq.jar:</td>
</tr>
<tr>
<td></td>
<td>install_dir[3]mqm/java/lib/connector.jar:</td>
</tr>
<tr>
<td></td>
<td>install_dir[3]mqm/java/samples/base:</td>
</tr>
<tr>
<td>iSeries and AS/400</td>
<td>CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar:</td>
</tr>
<tr>
<td></td>
<td>/QIBM/ProdData/mqm/java/lib/connector.jar:</td>
</tr>
<tr>
<td></td>
<td>/QIBM/ProdData/mqm/java/samples/base:</td>
</tr>
<tr>
<td>Linux</td>
<td>CLASSPATH=/opt/mqm/java/lib/com.ibm.mq.jar:</td>
</tr>
<tr>
<td></td>
<td>/opt/mqm/java/lib/connector.jar:</td>
</tr>
<tr>
<td></td>
<td>/opt/mqm/samp/java/base:</td>
</tr>
</tbody>
</table>

Notes:
1. mg_root_dir stands here for the directory used to install WebSphere MQ on Windows systems. This is normally C:\Program Files\IBM\WebSphere MQ\.
2. install_dir is the directory in which you installed the product.

To use WebSphere MQ JMS, you must include additional jar files in the classpath. These are listed in “Post installation setup” on page 25.

If there are existing applications with a dependency on the deprecated bindings package com.ibm.mqbind, you must also add the file com.ibm.mqbind.jar to your classpath.

You must update additional environment variables on some platforms, as shown in Table 5 on page 12.
Table 5. Environment variables for the product

<table>
<thead>
<tr>
<th>Platform</th>
<th>Environment variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>LIBPATH=/usr/mqm/java/lib</td>
</tr>
<tr>
<td>HP-UX</td>
<td>SHLIB_PATH=/opt/mqm/java/lib</td>
</tr>
<tr>
<td>Solaris</td>
<td>LD_LIBRARY_PATH=/opt/mqm/java/lib</td>
</tr>
<tr>
<td>Windows systems</td>
<td>PATH=install_dir\lib</td>
</tr>
<tr>
<td>z/OS and OS/390</td>
<td>LIBPATH=install_dir/mqm/java/lib</td>
</tr>
<tr>
<td>Linux</td>
<td>LD_LIBRARY_PATH=/opt/mqm/java/lib</td>
</tr>
</tbody>
</table>

**Note:** install_dir is the installation directory for the product

**Notes:**
1. To use WebSphere MQ Bindings for Java on OS/400, ensure that the library QMQMJAVA is in your library list.
2. Ensure that you append the WebSphere MQ variables and do not overwrite any of the existing system environment variables. If you overwrite existing system environment variables, the application might fail during compilation or at runtime.

**STEPLIB configuration on z/OS and OS/390**

On z/OS and OS/390, the STEPLIB used at runtime must contain the WebSphere MQ SCSQAUTH library. From UNIX System Services, you can add this using a line in your .profile as shown below, replacing th1qual with the high level data set qualifier that you chose when installing WebSphere MQ:

```bash
export STEPLIB=th1qual.SCSQAUTH:$STEPLIB
```

In other environments, you typically need to edit the startup JCL to include SCSQAUTH on the STEPLIB concatenation:

```bash
STEPLIB DD DSN=th1qual.SCSQAUTH,DISP=SHR
```

**Web server configuration**

If you install WebSphere MQ Java on a Web server, you can download and run WebSphere MQ Java applications on machines that do not have WebSphere MQ Java installed locally. To make the WebSphere MQ Java files accessible to your Web server, set up your Web server configuration to point to the directory where the client is installed. Consult your Web server documentation for details of how to configure this.

**Note:** On z/OS and OS/390, the installed classes do not support client connection and cannot be usefully downloaded to clients. However, jar files from another platform can be transferred to z/OS and OS/390 and served to clients.
Running WebSphere MQ Java applications under the Java 2 Security Manager

WebSphere MQ Java can run with the Java 2 Security Manager enabled. To successfully run applications with the Security Manager enabled, you must configure your JVM with a suitable policy definition file.

The simplest way to do this is to change the policy file supplied with the JRE. On most systems this file is stored in the path `lib/security/java.policy`, relative to your JRE directory. You can edit policy files using your preferred editor or the `policytool` program supplied with your JRE.

You need to give authority to the `com.ibm.mq.jar` and `com.ibm.mqjms.jar` files so that they can:
- Create sockets (in client mode)
- Load the native library (in bindings mode)
- Read various properties from the environment

The system property `os.name` must be available to the WebSphere MQ Java classes when running under the Java 2 Security Manager.

Here is an example of a policy file entry that allows WebSphere MQ Java to run successfully under the default security manager. Replace the string `/opt/mqm` in this example with the location where WebSphere MQ Java is installed on your system.

```java
grant codeBase "file:/opt/mqm/java/lib/com.ibm.mq.jar" {
    permission java.net.SocketPermission "*","connect";
    permission java.lang.RuntimePermission "loadLibrary.*";
};

grant codeBase "file:/opt/mqm/java/lib/com.ibm.mqjms.jar" {
    permission java.util.PropertyPermission "MQJMS_LOG_DIR","read";
    permission java.util.PropertyPermission "MQJMS_TRACE_LEVEL","read";
    permission java.util.PropertyPermission "MQJMS_TRACE_DIR","read";
    permission java.util.PropertyPermission "MQ_JAVA_INSTALL_PATH","read";
    permission java.util.PropertyPermission "file.separator","read";
    permission java.util.PropertyPermission "os.name","read";
    permission java.util.PropertyPermission "user.name","read";
    permission java.util.PropertyPermission "com.ibm.mq.jms.cleanup","read";
};
```

This example of a policy file enables the WebSphere MQ Java classes to work correctly under the security manager, but you might still need to enable your own code to run correctly before your applications will work.

The sample code shipped with WebSphere MQ Java has not been specifically enabled for use with the security manager; however the IVT tests run with the above policy file and the default security manager in place.
Chapter 3. Using WebSphere MQ classes for Java (WebSphere MQ base Java)

This chapter tells you how to:

- Configure your system to run the sample applet and application programs to verify your WebSphere MQ base Java installation.
- Modify the procedures to run your own programs.

Remember to check the README file installed with the WebSphere MQ Java code for later or more specific information for your environment.

The procedures depend on the connection option you want to use. Follow the instructions in the section that is appropriate for your requirements.

Configuring your queue manager to accept client connections

Use the following procedures to configure your queue manager to accept incoming connection requests from the clients.

TCP/IP client

1. Define a server connection channel using the following procedures:

   For the OS/400 platform:
   a. Start your queue manager by using the STRMQM command.
   b. Define a sample channel called JAVA.CHANNEL by issuing the following command:

      ```
      CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME) MCAUSERID(SOMEUSERID) TEXT('Sample channel for WebSphere MQ classes for Java')
      ```

      where QMGRNAME is the name of your queue manager, and SOMEUSERID is an OS/400 user ID with appropriate authority to the WebSphere MQ resources.

   For z/OS or OS/390 platforms:

      Note: You must have the Client attachment feature installed on your target queue manager in order to connect using TCP/IP.
      a. Start your queue manager by using the START QMGR command.
      b. Define a sample channel called JAVA.CHANNEL by issuing the following command:

      ```
      DEF CHL('JAVA.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) DESCR('Sample channel for WebSphere MQ classes for Java')
      ```

      For other platforms:

      a. Start your queue manager by using the strmqm command.
      b. Type the following command to start the runmqsc program:

      ```
      runmqsc [QMNAME]
      ```
      c. Define a sample channel called JAVA.CHANNEL by issuing the following command:
Using WebSphere MQ base Java

2. Start a listener program with the following commands:

For Windows NT, Windows 2000 operating systems:
Issue the command:
runmqslr -t tcp [-m QMNAME] -p 1414

Note: If you use the default queue manager, you can omit the -m option.

For UNIX operating systems:
Configure the inetd daemon, so that the inetd starts the WebSphere MQ channels. See [WebSphere MQ Clients] for instructions on how to do this.

For the OS/400 operating system:
Issue the command:
STRMQMLSR MQMNAME(QMGRNAME)

where QMGRNAME is the name of your queue manager.

For the z/OS or OS/390 operating system:

a. Ensure your channel initiator is started. If not, start it by issuing the START CHINIT command.
b. Start the listener by issuing the command START LISTENER TRPTYPE(TCP) PORT(1414)

Verifying with the sample application

An installation verification program, MQIVP, is supplied with WebSphere MQ base Java. You can use this application to test all the connection modes of WebSphere MQ base Java. The program prompts for a number of choices and other data to determine which connection mode you want to verify. Use the following procedure to verify your installation:

1. To test a client connection:
   a. Configure your queue manager, as described in "Configuring your queue manager to accept client connections" on page 15
   b. Carry out the rest of this procedure on the client machine.

To test a bindings connection, carry out the rest of this procedure on the WebSphere MQ server machine.

2. Change to your samples directory.
   See Table 3 on page 10 to find where this is.

3. Type:
   java MQIVP

The program tries to:
   a. Connect to, and disconnect from, the named queue manager.
   b. Open, put, get, and close the system default local queue.
   c. Return a message if the operations are successful.

4. At the prompt (1):
   • To use a TCP/IP connection, enter a WebSphere MQ server host name.
   • To use native connection (bindings mode), leave the field blank. (Do not enter a name.)
Installation verification program

Here is an example of the prompts and responses you might see. The actual prompts and your responses depend on your WebSphere MQ network.

Please enter the IP address of the MQ server : ipaddress\(^{(1)}\)
Please enter the port to connect to : (1414)\(^{(2)}\)
Please enter the server connection channel name : channelname\(^{(2)}\)
Please enter the queue manager name : qmname

Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager

Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...

Notes:
1. If you choose server connection, you do not see the prompts marked \(^{(2)}\).
2. On z/OS and OS/390, leave the field blank at prompt \(^{(1)}\).
3. On OS/400, you can run the command java MQIVP only from the Qshell interactive interface (the Qshell is option 30 of OS/400, 5769-SS1). Alternatively, you can run the application by using the CL command RUNJAVA CLASS(MQIVP).
4. To use the WebSphere MQ bindings for Java on OS/400, you must ensure that the library QMQJMJAVA is in your library list.

Running your own WebSphere MQ base Java programs

To run your own Java applets or applications, use the procedures described for the verification programs, substituting your application name in place of MQIVP.

For information on writing WebSphere MQ base Java applications and applets, see Part 2, “Programming with WebSphere MQ base Java,” on page 61.

Solving WebSphere MQ base Java problems

If a program does not complete successfully, run the installation verification applet or installation verification program, and follow the advice given in the diagnostic messages. Both of these programs are described in Chapter 3, “Using WebSphereMQ classes for Java (WebSphere MQ base Java),” on page 15.

If the problems continue and you need to contact the IBM service team, you might be asked to turn on the trace facility. Refer to the following sections for the appropriate procedures for your system.

Tracing the sample application

To trace the MQIVP program, enter the following:

```
java MQIVP -trace n
```

where \(n\) is a number between 1 and 5, depending on the level of detail required. (The greater the number, the more information is gathered.)

For more information about how to use trace, see “Tracing WebSphere MQ base Java programs” on page 94.
Error messages

Here are some of the more common error messages that you might see:

**Unable to identify local host IP address**
The server is not connected to the network.
Connect the server to the network and retry.

**MQRC_ADAPTER_CONN_LOAD_ERROR**
If you see this z/OS error, ensure that the WebSphere MQ SCSQANLE and SCSQAUTH datasets are in your STEPLIB statement.
Chapter 4. Using WebSphere MQ classes for Java Message Service (WebSphere MQ JMS)

This chapter tells you how to:
- Set up and use JMS Postcard
- Set up your system to use the test and sample programs
- Run the point-to-point Installation Verification Test (IVT) program to verify your WebSphere MQ classes for Java Message Service installation
- Run the sample publish/subscribe Installation Verification Test (PSIVT) program to verify your publish/subscribe installation
- Run your own programs

JMS Postcard

JMS Postcard is a simple way to do the following:
- Verify that you have successfully installed WebSphere MQ and WebSphere MQ JMS on one computer and, optionally, on others as well
- Introduce you to messaging

Note: JMS Postcard is not supported on WebSphere MQ for z/OS or WebSphere MQ for iSeries.

Setting up JMS Postcard

To use JMS Postcard, make sure that the Java Messaging feature of WebSphere MQ for Windows NT and Windows 2000 (WebSphere MQ JMS) is installed. You also need a working Java Runtime Environment (JRE) at Java 1.3 level.

Before you can successfully run the JMS Postcard application, define the environment variables CLASSPATH, LIBPATH, MQ_JAVA_INSTALL_PATH, and MQ_JAVA_DATA_PATH. On Windows systems these variables are set as part of the install process. On other platforms you must set them yourself. For more information about these variables, see "Environment variables" on page 10.

Many operations that the Postcard application carries out on your behalf require the user to be a member of the WebSphere MQ administrators group (mqm). If you are not a member of mqm, get a member of the mqm group to set up the default configuration on your behalf. See "JMS Postcard default configuration" on page 22.

Starting

To start the JMS Postcard application, run the postcard script. This is supplied in the java/bin directory of the WebSphere MQ installation.

The first time that you run JMS Postcard, it asks you to complete the default configuration, which sets up a suitable queue manager to act as mailbox. See "JMS Postcard default configuration" on page 22.

Whenever you start a Postcard application, you must sign on and enter a nickname. (There are advanced options available on the sign-on dialog, see "Sign-on advanced options" on page 20 for details).
**JMS Postcard**

**Sign-on**

The sign-on dialog has a check box labelled Advanced. Check this to see the extended dialog where you can choose which queue manager is used by the Postcard program.

**Notes:**

1. If you have no queue managers at all, or just the default configuration, the checkbox is disabled.
2. Depending on what queue managers and clusters you have, the checkbox and options are in one of various combinations of enabled, disabled, and preselected.

**Sign-on advanced options**

**Use default configuration as mailbox**

This is the easiest way to use JMS Postcard on one or several computers. Make sure that the default configuration is installed on all the computers, that one of them holds the repository, and that all the others use the first one as their repository; this puts them all in the same cluster.

**Choose queue manager as mailbox**

Use the drop-down list to choose any one of your local queue managers. If you want to send postcards between two queue managers (on one or more computers) this way, make sure that one of the following conditions is true:

- The queue managers are in the same cluster (for more information about clusters, see the *WebSphere MQ Queue Manager Clusters* book).
- There are explicit connections between the queue managers.

**Sending a postcard**

To send a postcard successfully, you need two instances of the Postcard application with different nicknames. For example, suppose you start the Postcard application and use the nickname Will, and then start it again using the nickname Tim. Will can send postcards to Tim and Tim can send postcards to Will.

If Will and Tim are connected to the same queue manager, see "Running JMS Postcard with one queue manager."

If Tim is on a different queue manager manager (on the same or a different computer from Will), see "Running JMS Postcard with two queue managers" on page 21.

When the postcard arrives successfully, you know that your WebSphere MQ installation and WebSphere MQ JMS are working correctly.

For an alternative way of verifying the installation of WebSphere MQ JMS, run the IVTRun application from the command line. See "Running the point-to-point IVT" on page 31 for more information about this.

**Running JMS Postcard with one queue manager**

If you have already started the Postcard application with a nickname, for example, Will, and you want to send a postcard to a second nickname on this computer, follow these steps:

1. Move the first Postcard (Will) to one side of your screen, then start a second Postcard by running the postcard shell script again.
2. Enter your second nickname, for example Tim.

3. On Will’s Postcard fill in the To field with your second nickname, Tim. (You can leave the On field empty and Postcard will fill it in for you, or you can type in the queue manager name that you see below the Message box after On).

4. Click in the Message box, type your message in, and click the Send button.

5. Look in Tim’s Postcard to see the message arrive, and double-click on it to see the postcard itself.

6. Try using Tim to send a message back to Will. You can do this by selecting the message that arrived in Tim’s list, and clicking the Reply button.

**Note:** See “JMS Postcard configuration” on page 22 for advice about configuration.

### Running JMS Postcard with two queue managers

If you have already started JMS Postcard with a nickname, for example Will, and you want to send a postcard to a second nickname on a second queue manager on this, or another, computer, follow these steps:

1. Start the second Postcard, choosing one of the following:
   - JMS Postcard
     - On this computer, run the postcard shell script again, then in the sign-on dialog check Advanced and select the second queue manager you want to use.
     - On another computer, run the postcard shell script; or, on Windows systems, open WebSphere MQ First Steps and click on JMS Postcard.
   - MQI Postcard on Windows systems:
     - either start from WebSphere MQ First Steps (to use the default configuration), or open the WebSphere MQ Explorer, right-click on the queue manager you want to use and click All Tasks->Start a Postcard...

2. When the sign-on dialog appears, enter your second nickname (for example, Tim).

3. In the Postcard application on Will’s computer, fill in the To field with your second nickname (Tim), and in the On field put the queue manager name of the second postcard where Tim is. If you don’t know this name, on Tim’s computer in the Postcard look below the Message box after On; alternatively if both queue managers are in the default configuration cluster, you can just type in the short TCP/IP name of Tim’s computer and Postcard builds that into the queue manager name in the same way that the task that creates the default configuration does.

4. Type your message, and click Send.
   - Look in Tim’s Postcard to see the message arrive, and double-click on it to see the postcard itself.

5. Try sending a message from Tim’s computer back to Will.
   - You can do this by selecting the message that arrived in Tim’s list, and clicking Reply.

**Note:** See “JMS Postcard configuration” on page 22.

See also “How JMS Postcard works” on page 22.
JMS Postcard configuration

The Postcard application needs a suitable queue manager to act as mailbox. See "JMS Postcard default configuration" for the easiest way to get one. You will be prompted to install this default configuration the first time you start the Postcard application (see “Starting” on page 19).

Instead of using the default configuration, you can also start the Postcard application using any other local queue manager.

If you want to send postcards to another computer, or to other queue managers, the default configuration must include the option of being joined in the same cluster. The other queue managers must either be in the same cluster or you must create a connection explicitly between them.

See also “How JMS Postcard works.”

JMS Postcard default configuration

Installing the default configuration creates a special queue manager (with queues and channels), and optionally joins it to a cluster, to enable you to use the JMS Postcard application to verify your installation and see messaging working.

On WebSphere MQ for Windows NT and Windows 2000, the Default Configuration Wizard automatically opens when JMS Postcard is started and the wizard has not already been run on this computer.

On platforms other than Windows systems, you can also run the Default Configuration script, provided that there are no existing queue managers on this computer. On Windows systems, run Default Configuration from First Steps.

Note: You must be a member of the WebSphere MQ administrators group (mqm) to complete default configuration successfully. If you are not a member of mqm, get a member of the mqm group to set up the default configuration on your behalf.

How JMS Postcard works

This section tells you how the JMS Postcard works, including:
• “Starting up”
• “Receiving messages” on page 23
• “Sending messages” on page 23
• “How the postcards get there” on page 23
• “Tidying up undeliverable messages” on page 24
• “Exchanging messages between different WebSphere MQ Postcard applications” on page 24
• “Customizing JMS Postcard” on page 24

Starting up

When JMS Postcard starts, it checks to see what queue managers exist on this computer, and initializes the sign-on dialog accordingly. If there are no queue managers at all, it prompts you to install the default configuration.

JMS Postcard uses the Java Message Service method queueConnectionFactory.createQueueConnection() to connect to the default queue manager.
Receiving messages
All the time JMS Postcard is running, it polls a queue called postcard for incoming messages from other Postcard applications. If there is no queue called postcard, JMS Postcard creates one.

When JMS Postcard starts running, it creates a Java Message Service QueueReceiver object for the local postcard queue, providing as a parameter a selector string that filters the messages to be received from the queue by the Correlation Identifier (CorrelId field). The selector string defines that the postcard client should only receive messages where the CorrelId field matches the nickname of the user. The words from the message data are then presented in the JMS Postcard window.

Sending messages
If you did not enter a computer name in the On: field, JMS Postcard assumes that the recipient is on the same queue manager.

If you entered a name, JMS Postcard checks for the existence of a queue manager with this name, first using the exact name supplied, and then using a prefix in the same format as that created by the default configuration.

In both cases, it issues a session.createQueue('postcard'), and sets the base queue manager name to the string supplied.

Finally, it builds a JMS BytesMessage from your nickname and the words you typed in, and runs queueSender.send(theMessage) to put the message onto the queue.

How the postcards get there
When other instances of Postcard on this computer use the same queue manager and queue, the messages are being put and got from the one queue. This does, however, verify that the WebSphere MQ code installed on this computer is configured and working correctly.

JMS Postcard can only send to another queue manager if a connection to that queue manager exists. This connection exists because either both queue managers are members of the same cluster, or you have explicitly created a connection yourself. JMS Postcard can therefore assume that it can connect to the queue manager, and connects to it, opens the queue, and puts a message, as already described, leaving all the work of getting the message there to the WebSphere MQ cluster code. In other words, JMS Postcard uses only one piece of code for putting the message, and does not need to know whether the message is going to another computer.

In JMS Postcard, when session.createSender('postcard') is called, the cluster code checks the repository to find the other queue manager, and to check that the queue exists, and throws an exception if this was not possible for any reason.

When queueSender.send(theMessage) is called, the cluster code opens a channel to the other queue manager (creating it if necessary) and sends the message.

Discard the channel afterwards, if the cluster optimizing code does not need it. If the queue managers are on different computers, that is all handled by the cluster code.
**Tidying up undeliverable messages**

If you sent a postcard message to John, but never ran a Postcard application with the nickname John, the message would sit on the queue for ever. To prevent this, JMS Postcard sets the Message Lifetime (Expiry) field in the Message Descriptor (MQMD) to 48 hours. After that time, the message is discarded, wherever it may be (possibly even still in transmission).

**Exchanging messages between different WebSphere MQ Postcard applications**

You can exchange messages between all the different types of Postcard application as follows:

- **MQI Postcard** on WebSphere MQ for Windows NT and Windows 2000.
- **JMS Postcard** on Windows systems and other operating systems such as UNIX.
- **MQSeries Postcard** on previous versions of MQSeries for Windows, with the exception that it cannot receive messages from JMS Postcard.
- **MQ Everyplace Postcard** on WebSphere MQ Everyplace on pervasive devices.

For this, a connection must be explicitly set up between the queue managers. See the WebSphere MQ Everyplace product documentation for further information.

**Customizing JMS Postcard**

Normally JMS Postcard uses standard Java Swing settings for font size and background color. But if it detects a postcard.ini file on startup, JMS Postcard uses settings specified in this file instead. You can also change the trace setting.

Edit the sample file postcard.ini in the bin directory of the WebSphere MQ classes for Java installation and set your preferred settings for font size, and screen foreground and background colors.

**Note:** The precise use of upper and lower case letters in the keywords, as in the following examples, must be strictly observed when you set these properties.

**Setting screen colors**

By setting the Background and Foreground properties, you can change the background and foreground colors of controls used in the Postcard application.

```
Background=000000
Foreground=FFFFFF
```

This example selects white text on a black background. The values represent intensity levels for red, green, and blue colors using a hexadecimal scale from 00 to FF. Other examples of colors are FF0000 (bright red), 00FF00 (bright green) and 0000FF (bright blue).

**Setting font size**

```
MinimumFont=20
```

This example selects a minimum font size of 20 points. Any value smaller than 13 is ignored.

**Using an external browser for online help**

```
WebBrowser=nautilus
```

This setting is only applicable on non-Windows systems. The internal browser used for displaying online help information cannot be customized. This setting allows you to identify an alternative browser.
Tracing the Postcard application

Trace=1

Set this to start trace output. Note that the trace output is sent to the trc subdirectory of the directory defined by the MQ_JAVA_DATA_PATH system environment variable. If the application cannot write to this directory, trace output is directed to the system console.

You can also use the MQJMS_TRACE_LEVEL parameter on the java command line to start tracing. See “Tracing programs” on page 38 for more about tracing applications.

Post installation setup

Note: Remember to check the README file installed with the WebSphere MQ Java programs for information that may supersede this book.

To make all the necessary resources available to WebSphere MQ JMS programs, you need to update the following system variables:

Classpath

Successful operation of JMS programs requires a number of Java packages to be available to the JVM. You must specify these on the classpath after you have obtained and installed the necessary packages.

Add the following .jar files to the classpath:

- com.ibm.mq.jar
- com.ibm.mqjms.jar
- connector.jar
- jms.jar
- jndi.jar
- jta.jar
- providerutil.jar
- fscontext.jar
- ldap.jar

Notes:

1. For z/OS and OS/390, use ibmjndi.jar and jndi.jar from /usr/lpp/ldap/lib instead of jndi.jar and ldap.jar. These files are supplied with the operating system.
2. Include the java/lib directory itself in the classpath to access the properties files used by the base Java API.
3. Include providerutil.jar, jndi.jar, and either ldap.jar or fscontext.jar if you need to access a JNDI namespace.
4. In certain environments, typically J2EE application servers, classes contained in these jars are provided by the environment. In these circumstances, use the classes provided by the environment instead of those provided with WebSphere MQ.

Environment variables

There are a number of scripts in the bin subdirectory of the WebSphere MQ JMS installation. These are for use as convenient shortcuts for a number of common actions. Many of these scripts assume that the environment variables MQ_JAVA_INSTALL_PATH and MQ_JAVA_DATA_PATH are...
Setup

defined, pointing to the directory in which WebSphere MQ JMS is installed and a directory for log and trace output, respectively. If you do not set these variables, you must edit the scripts in the bin directory accordingly.

On Windows NT, you can set the classpath and other environment variables by using the Environment tab of System Properties. On Windows 2000 and Windows XP, Environment is a button on the Advanced tab of System Properties. On UNIX, these are normally set from each user’s logon scripts. On any platform, you can use scripts to maintain different classpaths and other environment variables for different projects.

**Note:** If you are migrating from the SupportPac MA88, be aware that the connector.jar is now packaged in the java/lib directory with the other jar files, with the following consequences:

- You need an entry for connector.jar in the classpath, as explained above.
- If you have previously implemented your own ConnectionManagers, as described in “Supplying your own ConnectionManager” on page 85, you must replace references to com.ibm.mq.resource and com.ibm.mq.resource.spi with references to javax.resource and javax.resource.spi respectively.

### Additional setup for publish/subscribe mode

Before you can use the WebSphere MQ JMS implementation of JMS publish/subscribe, some additional setup is required:

- Ensure that you have access to a publish/subscribe broker.
- Ensure that the broker is running.
- Create the WebSphere MQ JMS system queues.

This step is not required for direct connection across a TCP/IP socket to a WebSphere MQ Event Broker.

You also need to know publish/subscribe concepts as discussed in Chapter 11, “Writing WebSphere MQ JMS publish/subscribe applications,” on page 213.

**Ensure that you have access to a publish/subscribe broker**

With WebSphere MQ JMS you have the choice of three brokers:

- WebSphere MQ with SupportPac MA0C (also known as MQSeries Publish/Subscribe)
- WebSphere MQ Integrator V2
- WebSphere MQ Event Broker

Differences between these brokers are discussed in Chapter 11, “Writing WebSphere MQ JMS publish/subscribe applications,” on page 213. Read the documentation for each broker for installation and configuration instructions.

**Note:** To use broker-based subscription stores, you must use SupportPac MA0C or the WebSphere MQ Event Broker. No other combination of queue manager and broker supports this option. For more information about subscription stores, see “Subscription stores” on page 227. For information specific to JMS 1.1, see “Subscription stores” on page 246.
Ensure that the broker is running

MQSeries Publish/Subscribe
To verify that the broker is installed and running, use the command:

dsmqbrk -m MY.QUEUE.MANAGER

where MY.QUEUE.MANAGER is the name of the queue manager on which the broker is running. If the broker is running, a message similar to the following is displayed:

WebSphere MQ message broker for queue manager MY.QUEUE.MANAGER running.

If the operating system reports that it cannot run the dsmqbrk command, ensure that the MQSeries Publish/Subscribe broker is installed properly.

If the operating system reports that the broker is not active, start it using the command:

strmqbrk -m MY.QUEUE.MANAGER

WebSphere MQ Integrator V2
To verify that the broker provided in WebSphere MQ Integrator V2 is installed and running, refer to the product documentation.

The command to start the broker in WebSphere MQ Integrator V2 is:

mqsistart MYBROKER

where MYBROKER is the name of the broker.

WebSphere MQ Event Broker
To verify that the broker provided in WebSphere MQ Event Broker is installed and running, refer to the product documentation.

The command to start the broker in WebSphere MQ Event Broker is:

wmqpsstart MYBROKER

where MYBROKER is the name of the broker.

Create the WebSphere MQ JMS system queues
This does not apply if you use a direct connection across TCP/IP to WebSphere MQ Event Broker.

For a publish/subscribe implementation to work correctly, you must create a number of system queues. A script is supplied, in the bin subdirectory of the WebSphere MQ JMS installation, to assist with this task. To use the script, enter the following commands:
Setup for publish/subscribe

For iSeries and AS/400:

1. Copy the script from the integrated file system to a native file system library using a command similar to:
   
   COPYRSTMF FROMSTMF('/QIBM/ProdData/mqm/java/bin/MQJMS_PSQ.mqsc')
   TOBR('/QSYS.LIB/QGPL.LIB/QCLSRC.FILE/MQJMS_PSQ.MBR')

2. Call the script file using STRMQMMQSC:
   
   STRMQMMQSC SRCMBR(MQJMS_PSQ) SRCFILE(QGPL/QCLSRC)

For z/OS and OS/390:

1. Copy the script from the HFS into a PDS using a TSO command similar to:
   
   OGET '/usr/lpp/mqm/java/bin/MQJMS_PSQ.mqsc' 'USERID.MQSC(MQJMSPSQ)'

   The PDS should be of fixed-block format with a record length of 80.

2. Either use the CSQUTIL application to execute this command script, or add the script to the CSQINP2 DD concatenation in your queue manager’s started task JCL. In either case, refer to the WebSphere MQ for z/OS System Setup Guide and the WebSphere MQ for z/OS System Administration Guide for further details.

For other platforms:

   runmqsc MY.QUEUE.MANAGER < MQJMS_PSQ.mqsc

If an error occurs, check that you typed the queue manager name correctly and that the queue manager is running.

For a broker running on a remote queue manager

For operation with a broker running on a remote queue manager, further setup is required.

1. Define a transmission queue on the remote queue manager with a queue name matching the local queue manager. These names must match for correct routing of messages by WebSphere MQ.

2. Define a sender channel on the remote queue manager and a receiver channel on the local queue manager. The sender channel should use the transmission queue defined in step 1.

3. Set up the local queue manager for communication with the remote broker:
   a. Define a local transmission queue with the same name as the queue manager running the remote broker.
   b. Define local sender and remote receiver channels to the remote broker queue manager. The sender channel must use the transmission queue defined in step 3a.

4. To operate the remote broker, take the following steps:
   a. Start the remote broker queue manager.
   b. Start a listener for the remote broker queue manager (TCP/IP channels).
   c. Start the sender and receiver channels to the local queue manager.
   d. Start the broker on the remote queue manager.

   An example command is
   
   strmqbrk -m MyBrokerMgr

5. To operate the local queue manager to communicate with the remote broker, take the following steps:
a. Start the local queue manager.
b. Start a listener for the local queue manager.
c. Start the sender and receiver channels to the remote broker queue manager.

Queues that require authorization for non-privileged users

Non-privileged users need authorization granted to access the queues used by JMS. For details about access control in WebSphere MQ, see the chapter about protecting WebSphere MQ objects in the WebSphere MQ System Administration Guide.

For JMS point-to-point mode, the access control issues are similar to those for the WebSphere MQ classes for Java:

- Queues that are used by QueueSender need put authority.
- Queues that are used by QueueReceivers and QueueBrowsers need get, inq, and browse authorities.
- The QueueSession.createTemporaryQueue method needs access to the model queue that is defined in the QueueConnectionFactory temporaryModel field (by default this is SYSTEM.DEFAULT.MODEL.QUEUE).

For JMS publish/subscribe mode, the following system queues are used:

- SYSTEM.JMS.ADMIN.QUEUE
- SYSTEM.JMS.REPORT.QUEUE
- SYSTEM.JMS.MODEL.QUEUE
- SYSTEM.JMS.PS.STATUS.QUEUE
- SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.SUBSCRIBER.QUEUE
- SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
- SYSTEM.BROKER.CONTROL.QUEUE

Also, any application that publishes messages needs access to the STREAM queue that is specified in the topic connection factory being used. The default value for this is SYSTEM.BROKER.DEFAULT.STREAM.

If you use ConnectionConsumer, additional authorization might be needed. Queues to be read by the ConnectionConsumer must have get, inq and browse authorities. The system dead-letter queue, and any backout-requeue queue or report queue used by the ConnectionConsumer must have put and passall authorities.

Using the sample JMS applet to verify the TCP/IP client

WebSphere MQ JMS includes an installation verification applet, test.html. You can use the applet to verify the TCP/IP connected client mode of WebSphere MQ JMS except on the z/OS and OS/390 platform, where the TCP/IP connected client mode is not supported.

The standard security settings for applets in Java 1.2 and higher require that all referenced classes are loaded from the same location as the applet you want to run. For information on how to ensure that applets using WebSphere MQ JMS work, see Appendix E, “Using WebSphere MQ Java in applets with Java 1.2 or later,” on page 481.
Verifying TCP/IP client

The applet connects to a given queue manager, exercises all the WebSphere MQ calls, and produces diagnostic messages if there are any failures. If the applet does not complete successfully, follow the advice given in the diagnostic messages and run the applet again.

Using the sample applet with OS/400

The OS/400 operating system does not have a native Graphical User Interface (GUI). To run the sample applet, you need to use the Remote Abstract Window Toolkit for Java (AWT), or the Class Broker for Java (CBJ), on graphics capable hardware.

Running the sample applet

First make sure that your queue manager can accept client connections. For details of this, see “Configuring your queue manager to accept client connections” on page 15.

There are different ways of running the JMS sample applet. Each has slightly different properties because of the security restrictions on applets imposed by the Java virtual machine.

Normal Java security settings cause the appletviewer or browser to ignore your system CLASSPATH, so the WebSphere MQ base Java and WebSphere MQ JMS libraries must be present in the same location as the applet class file. For further details of applets and security settings, see Appendix E, “Using WebSphere MQ Java in applets with Java 1.2 or later,” on page 481.

Running from a web server (in appletviewer or in a browser):
Invoke the applet using a command line like the following:
appletviewer http://<web.server.host/jmsapplet>/test.html

or by pointing your Java 1.3 enabled browser at this Web page. Change the string <web.server.host/jmsapplet> as appropriate to the URL of the Web server you are using.

Running in appletviewer from the local machine:
Invoke the applet using a command line like the following:
appletviewer test.html

Remember that the WebSphere MQ base Java and WebSphere MQ JMS libraries must be present in the same local directory as the applet class file. Also, in this case, the applet might connect only to queue managers on the local machine.

Running the applet as an application:

Compile the applet using the command:
javac JMSTestApplet.java

Then run the applet using the command:
java JMSTestApplet

The JMS sample applet contains a main method that allows the applet to run as a standalone Java application.

This option requires the WebSphere MQ base Java and WebSphere MQ JMS libraries to be present in the system CLASSPATH, as for your own
Running Point-to-point

WebSphere MQ JMS applications. It allows you to connect to any host and queue manager to which you have TCP/IP access.

**Tracing the sample as an application**
To trace the sample as an application, alter the command line parameters as shown below, in the same way as you would trace your own JMS applications:
```shell
java -DMQJMS_TRACE_LEVEL=on JMSTestApplet
```
More details can be found in “Tracing programs” on page 38.

### Running the point-to-point IVT

This section describes the point-to-point installation verification test program (IVT) that is supplied with WebSphere MQ JMS.

The IVT verifies the installation by connecting to the default queue manager on the local machine, using the WebSphere MQ JMS in bindings mode. It then sends a message to the SYSTEM.DEFAULT.LOCAL.QUEUE queue and reads it back again.

You can run the program in one of two possible modes.

**With JNDI lookup of administered objects**
JNDI mode forces the program to obtain its administered objects from a JNDI namespace, which is the expected operation of JMS client applications. (See “Administering JMS objects” on page 45 for a description of administered objects). This invocation method has the same prerequisites as the administration tool (see Chapter 5, “Using the WebSphere MQ JMS administration tool,” on page 41).

**Without JNDI lookup of administered objects**
If you do not want to use JNDI, you can create the administered objects at runtime by running the IVT in non-JNDI mode. Because a JNDI-based repository is relatively complex to set up, run the IVT first without JNDI.

### Point-to-point verification without JNDI

A script, named IVTRun on UNIX, or IVTRun.bat on Windows systems, is provided to run the IVT. This file is installed in the bin subdirectory of the installation.

To run the test without JNDI, issue the following command:
```bash
IVTRun [-t] -nojndi [-m <qmgr>]
```

For client mode, to run the test without JNDI, issue the following command:
```bash
IVTRun [-t] -nojndi -client -m <qmgr> -host <hostname> [-port <port>]
[-channel <channel>]
```

where:

- **-t** turns tracing on (by default, tracing is off)
- **qmgr** is the name of the queue manager to which you want to connect
- **hostname** is the host on which the queue manager is running
- **port** is the TCP/IP port on which the queue manager’s listener is running (default 1414)
- **channel** is the client connection channel (default SYSTEM.DEF.SVRCONN)

If the test completes successfully, you should see output similar to the following:
Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:414d512041534341524920202020207cce883c03300020
JMSTimestamp: 1016124013892
JMSCorrelationID: null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_PutDate:20020314
JMSXAppID:java
JMS_IBM_Format:MQSTR
JMS_IBM_PutAppIType:6
JMS_IBM_MsgType:8
JMSXUserID:parkiw
JMS_IBM_PutTime:16401390
JMSDeliveryCount:1

A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished

Point-to-point verification with JNDI

To run the IVT with JNDI, the LDAP server must be running and must be configured to accept Java objects. If the following message occurs, it indicates that there is a connection to the LDAP server, but that the server is not correctly configured:

Unable to bind to object

This message means that either the server is not storing Java objects, or the permissions on the objects or the suffix are not correct. See "Checking your LDAP server configuration" on page 463.

Also, the following administered objects must be retrievable from a JNDI namespace:

- MQQueueConnectionFactory
- MQQueue

A script, named IVTSetup on UNIX, or IVTSetup.bat on Windows systems, is provided to create these objects automatically. Enter the command:
IVTSetup
The script invokes the WebSphere MQ JMS Administration tool (see Chapter 5, “Using the WebSphere MQ JMS administration tool,” on page 41) and creates the objects in a JNDI namespace.

The MQQueueConnectionFactory is bound under the name ivtQCF (for LDAP, cn=ivtQCF). All the properties are default values:
- TRANSPORT(BIND)
- PORT(1414)
- HOSTNAME(localhost)
- CHANNEL(SYSTEM.DEF.SVRCONN)
- VERSION(1)
- CCSID(819)
- TEMPMODEL(SYSTEM.DEFAULT.MODEL.QUEUE)
- QMANAGER()

The MQQueue is bound under the name ivtQ (cn=ivtQ). The value of the QUEUE property becomes QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE). All other properties have default values:
- PERSISTENCE(APP)
- QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE)
- EXPIRY(APP)
- TARGCLIENT(JMS)
- ENCODING(NATIVE)
- VERSION(1)
- CCSID(1208)
- PRIORITY(APP)
- QMANAGER()

Once the administered objects are created in the JNDI namespace, run the IVTRun (IVTRun.bat on Windows systems) script using the following command:
IVTRun [-t] -url "<providerURL>" [-icf <initCtxFact> ]

where:
- -t turns tracing on (by default, tracing is off)
- providerURL

**Note:** Enclose the *providerURL* string in quotation marks ("). This is the JNDI location of the administered objects. If the default initial context factory is in use, this is an LDAP URL of the form:
"ldap://hostname.company.com/contextName"

If a file system service provider is used, (see initCtxFact below), the URL is of the form:
"file://directorySpec"

- initCtxFact is the classname of the initial context factory. The default is for an LDAP service provider, and has the value:
  com.sun.jndi.ldap.LdapCtxFactory

  If a file system service provider is used, set this parameter to:
  com.sun.jndi.fscontext.RefsContextFactory

If the test completes successfully, the output is similar to the non-JNDI output, except that the create QueueConnectionFactory and Queue lines indicate retrieval of the object from JNDI. The following shows an example.
Websphere MQ classes for Java™ Message Service 5.300

Installation Verification Test

Using administered objects, please ensure that these are available

Retrieving a QueueConnectionFactory from JNDI
Creating a Connection
Creating a Session
Retrieving a Queue from JNDI
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
   JMSType: null
   ...
   ...

Although not strictly necessary, it is good practice to remove objects that are created by the IVTSetup script from the JNDI namespace. A script called IVTTidy (IVTTidy.bat on Windows systems) is provided for this purpose.

IVT error recovery

If the test is not successful, note the following:

- For help with any error messages involving the classpath, check that your classpath is set correctly, as described in “Post installation setup” on page 25.
- The IVT might fail with a message failed to create MQQueueManager, with an additional message including the number 2059. This indicates that WebSphere MQ failed to connect to the default local queue manager on the machine on which you ran the IVT. Check that the queue manager is running, and that it is marked as the default queue manager.
- A message failed to open MQ queue indicates that WebSphere MQ connected to the default queue manager, but could not open the SYSTEM.DEFAULT.LOCAL.QUEUE. This might indicate that either the queue does not exist on your default queue manager, or that the queue is not enabled for PUT and GET. Add or enable the queue for the duration of the test.

Table 6 lists the classes that are tested by IVT, and the package that they come from:

Table 6. Classes that are tested by IVT

<table>
<thead>
<tr>
<th>Class</th>
<th>Jar file</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebSphere MQ JMS classes</td>
<td>com.ibm.mq.jms.jar</td>
</tr>
<tr>
<td>com.ibm.mq.MQMessage</td>
<td>com.ibm.mq.jar</td>
</tr>
<tr>
<td>javax.jms.Message</td>
<td>jms.jar</td>
</tr>
<tr>
<td>javax.naming.InitialContext</td>
<td>jndi.jar</td>
</tr>
<tr>
<td>javax.resource.cci.Connection</td>
<td>connector.jar</td>
</tr>
<tr>
<td>javax.transaction.xa.XAException</td>
<td>jta.jar</td>
</tr>
<tr>
<td>com/sun/jndi/toolkit/ComponentDirContext</td>
<td>providerutil.jar</td>
</tr>
<tr>
<td>com.sun.jndi.ldap.LdapCtxtFactory</td>
<td>ldap.jar</td>
</tr>
</tbody>
</table>
The publish/subscribe installation verification test

The publish/subscribe installation verification test (PSIVT) program is supplied only in compiled form. It is in the com.ibm.mq.jms package.

The test requires a broker such as the MQSeries Publish/Subscribe broker (SupportPac MA0C) or WebSphere MQ Integrator V2 to be installed and running.

The PSIVT attempts to:
1. Create a publisher, p, publishing on the topic MQJMS/PSIVT/Information
2. Create a subscriber, s, subscribing on the topic MQJMS/PSIVT/Information
3. Use p to publish a simple text message
4. Use s to receive a message waiting on its input queue

When you run the PSIVT, the publisher publishes the message, and the subscriber receives and displays the message. The publisher publishes to the broker's default stream. The subscriber is non-durable, does not perform message selection, and accepts messages from local connections. It performs a synchronous receive, waiting a maximum of 5 seconds for a message to arrive.

You can run the PSIVT, like the IVT, in either JNDI mode or standalone mode. JNDI mode uses JNDI to retrieve a TopicConnectionFactory and a Topic from a JNDI namespace. If JNDI is not used, these objects are created at runtime.

Publish/subscribe verification without JNDI

A script named PSIVTRun (PSIVTRun.bat on Windows systems) is provided to run PSIVT. The file is in the bin subdirectory of the installation.

To run the test without JNDI, issue the following command:
PSIVTRun -nojndi [-m <qmgr>] [-bqm <broker>] [-t]

For client mode, to run the test without JNDI, issue the following command:
PSIVTRun -nojndi -client -m <qmgr> -host <hostname> [-port <port>] [-channel <channel>] [-bqm <broker>] [-t]

where:
- **nojndi** indicates no JNDI lookup of the administered objects
- **qmgr** is the name of the queue manager to which you wish to connect
- **hostname** is the host on which the queue manager is running
- **port** is the TCP/IP port on which the queue manager's listener is running (default 1414)
- **channel** is the client connection channel (default SYSTEM.DEFSVRCONN)
- **broker** is the name of the remote queue manager on which the broker is running. If this is not specified, the value used for qmgr is assumed.
- **-t** turns tracing on (default is off)

If the test completes successfully, output is similar to the following:
Creating a Connection
Creating a TopicConnectionFactory
Creating a Session
Creating a Topic
Creating a TopicPublisher
Creating a TopicSubscriber
Creating a TextMessage
Adding text
Publishing the message to topic://MQJMS/PSIVT/Information
Waiting for a message to arrive [5 secs max]...

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:414d51204153434152492020202020207cce883c19230020
JMSTimestamp: 1016124933637
JMSCorrelationID:ID:414d51204153434152492020202020207cce883c09320020
JMSDestination: topic://MQJMS/PSIVT/Information
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_PutDate:20020314
JMSAppID:ASCARI
JMS_IBM_Format:MQSTR
JMS_IBM_MsgType:8
JMSXUserID:parkiw
JMS_IBM_PutTime:16553367
JMSXDeliveryCount:1
A simple text message from the MQJMS/PSIVT program
Reply string equals original string
Closing TopicSubscriber
Closing TopicPublisher
Closing Session
Closing Connection
PSIVT finished

Publish/subscribe verification with JNDI

To run the PSIVT in JNDI mode, two administered objects must be retrievable from
a JNDI namespace:
• A TopicConnectionFactory bound under the name ivtTCF
• A Topic bound under the name ivtT

You can define these objects by using the WebSphere MQ JMS Administration Tool
(see Chapter 5, “Using the WebSphere MQ JMS administration tool,” on page 41)
and using the following commands:
DEFINE TCF(ivtTCF)

This command defines the TopicConnectionFactory.
DEFINE T(ivtT) TOPIC(MQJMS/PSIVT/Information)

This command defines the Topic.

These definitions assume that a default queue manager, on which the broker is
running, is available. For details on configuring these objects to use a non-default
queue manager, see “Administering JMS objects” on page 45. These objects must resi
de in a context pointed to by the -url command-line parameter described
below.
To run the test in JNDI mode, enter the following command:

```bash
PSIVTRun [-t] -url "<providerURL>" [-icf <initCtxFact> ]
```

where:

- `-t` means turn tracing on (by default, tracing is off)

**providerURL**

Note: Enclose the `providerURL` string in quotation marks (`"`). This is the JNDI location of the administered objects. If the default initial context factory is in use, this is an LDAP URL of the form:

```
"ldap://hostname.company.com/contextName"
```

If a file system service provider is used, (see `initCtxFact` below), the URL is of the form:

```
"file://directorySpec"
```

**initCtxFact** is the classname of the initial context factory. The default is for an LDAP service provider, and has the value:

```
com.sun.jndi.ldap.LdapCtxFactory
```

If a file system service provider is used, set this parameter to:

```
com.sun.jndi.fscontext.RefFSContextFactory
```

If the test completes successfully, output is similar to the non-JNDI output, except that the create QueueConnectionFactory and Queue lines indicate retrieval of the object from JNDI.

**PSIVT error recovery**

If the test is not successful, note the following:

- The following message:
  
  *** No broker response. Please ensure broker is running. ***

  indicates that the broker is installed on the target queue manager, but its control queue contains some outstanding messages. For instructions on how to start it, see [Additional setup for publish/subscribe mode](#) on page 26.

- If the following message is displayed:
  
  Unable to connect to queue manager: <default>

  ensure that your WebSphere MQ system has configured a default queue manager.

- If the following message is displayed:
  
  Unable to connect to queue manager: ...

  ensure that the administered TopicConnectionFactory that the PSIVT uses is configured with a valid queue manager name. Alternatively, if you used the `-nojndi` option, ensure that you supplied a valid queue manager (using the `-m` option).

- If the following message is displayed:
  
  Unable to access broker control queue on queue manager: ...
  Please ensure the broker is installed on this queue manager
ensure that the administered TopicConnectionFactory that the PSIVT uses is configured with the name of the queue manager on which the broker is installed. If you used the `-nojndi` option, ensure that you supplied a queue manager name (using the `-m` option).

Running your own WebSphere MQ JMS programs

For information about writing your own WebSphere MQ JMS programs, see “Programming with WebSphere MQ JMS,” on page 195.

WebSphere MQ JMS includes a utility file, `runjms` (runjms.bat on Windows systems), to help you to run the supplied programs and programs that you have written.

The utility provides default locations for the trace and log files, and enables you to add any application runtime parameters that your application needs. The supplied script assumes that the environment variable `MQ_JAVA_INSTALL_PATH` is set to the directory in which WebSphere MQ JMS is installed. The script also assumes that the subdirectories trace and log within the directory pointed to by `MQ_JAVA_DATA_PATH` are used for trace and log output, respectively.

Use the following command to run your application:

`runjms <classname of application> [application-specific arguments]`

Solving problems

If a program does not complete successfully, run the installation verification program, which is described in “Running the point-to-point IVT” on page 31, and follow the advice given in the diagnostic messages.

Tracing programs

The WebSphere MQ JMS trace facility is provided to help IBM staff to diagnose customer problems.

Trace is disabled by default, because the output rapidly becomes large, and is unlikely to be of use in normal circumstances.

If you are asked to provide trace output, enable it by setting the Java property `MQJMS_TRACE_LEVEL` to one of the following values:

- **on** — traces WebSphere MQ JMS calls only
- **base** — traces both WebSphere MQ JMS calls and the underlying WebSphere MQ base Java calls

For example:

```
java -DMQJMS_TRACE_LEVEL=base MyJMSProg
```

To disable trace, set `MQJMS_TRACE_LEVEL` to `off`.

By default, trace is output to a file named `mqjms.trc` in the current working directory. You can redirect it to a different directory by using the Java property `MQJMS_TRACE_DIR`.

For example:

```
java -DMQJMS_TRACE_LEVEL=base -DMQJMS_TRACE_DIR=/somepath/tracedir MyJMSProg
```
The runjms utility script sets these properties by using the environment variables MQJMS_TRACE_LEVEL and MQ_JAVA_DATA_PATH, as follows:

```java
java -DMQJMS_LOG_DIR=%MQ_JAVA_DATA_PATH%\log
-DMQJMS_TRACE_DIR=%MQ_JAVA_DATA_PATH%\trace
-DMQJMS_TRACE_LEVEL=%MQJMS_TRACE_LEVEL%
```

This is the default; change it as required.

**Logging**

The WebSphere MQ JMS log facility is provided to report serious problems, particularly those that might indicate configuration errors rather than programming errors. By default, log output is sent to the System.err stream, which usually appears on the stderr of the console in which the JVM is run.

You can redirect the output to a file by using a Java property that specifies the new location, for example:

```java
java -DMQJMS_LOG_DIR=/mydir/forlogs MyJMSProg
```

The utility script runjms, in the bin directory of the WebSphere MQ JMS installation, sets this property to:

```
<MQ_JAVA_DATA_PATH>/log
```

where MQ_JAVA_DATA_PATH is set, on Windows systems, to the path to your WebSphere MQ Java installation. On other platforms you need to set this environment variable.

When the log is redirected to a file, it is output in a binary form. To view the log, the utility formatLog (formatLog.bat on Windows systems) is provided, which converts the file to plain text format. The utility is stored in the bin directory of your WebSphere MQ JMS installation. Run the conversion as follows:

```bash
formatLog <inputfile> <outputfile>
```
Logging
Chapter 5. Using the WebSphere MQ JMS administration tool

The administration tool enables administrators to define the properties of eight types of WebSphere MQ JMS object and to store them within a JNDI namespace. Then, JMS clients can use JNDI to retrieve these administered objects from the namespace and use them.

The JMS objects that you can administer by using the tool are:

- MQConnectionFactory (JMS 1.1 only)
- MQQueueConnectionFactory
- MQTopicConnectionFactory
- MQQueue
- MQTopic
- MQXAConnectionFactory (JMS 1.1 only)
- MQXAQueueConnectionFactory
- MQXATopicConnectionFactory
- JMSWrapXAQueueConnectionFactory
- JMSWrapXATopicConnectionFactory

For details about these objects, refer to “Administering JMS objects” on page 45.

Note: JMSWrapXAQueueConnectionFactory and JMSWrapXATopicConnectionFactory are classes that are specific to WebSphere Application Server. They are contained in the package com.ibm.ejs.jms.mq.

The tool also allows administrators to manipulate directory namespace subcontexts within the JNDI. See “Manipulating subcontexts” on page 45.

Invoking the administration tool

The administration tool has a command line interface. You can use this interactively, or use it to start a batch process. The interactive mode provides a command prompt where you can enter administration commands. In the batch mode, the command to start the tool includes the name of a file that contains an administration command script.

To start the tool in interactive mode, enter the command:

JMSAdmin [-t] [-v] [-cfg config_filename]

where:

- `-t` Enables trace (default is trace off)
- `-v` Produces verbose output (default is terse output)
- `-cfg config_filename` Names an alternative configuration file (see “Configuration” on page 42)

A command prompt is displayed, which indicates that the tool is ready to accept administration commands. This prompt initially appears as:

InitCtx>
indicating that the current context (that is, the JNDI context to which all naming and directory operations currently refer) is the initial context defined in the PROVIDER_URL configuration parameter (see "Configuration").

As you traverse the directory namespace, the prompt changes to reflect this, so that the prompt always displays the current context.

To start the tool in batch mode, enter the command:

```
JMSAdmin <test.scp
```

where `test.scp` is a script file that contains administration commands (see "Administration commands" on page 44). The last command in the file must be the `END` command.

### Configuration

Configure the administration tool with values for the following three properties:

**INITIAL_CONTEXT_FACTORY**

The service provider that the tool uses. There are three explicitly supported values for this property:

- `com.sun.jndi.ldap.LdapCtxFactory` (for LDAP)
- `com.sun.jndi.fscontext.RefFSContextFactory` (for file system context)
- `com.ibm.websphere.naming.WsnInitialContextFactory` (to work with WebSphere Application Server’s CosNaming repository)

On z/OS and OS/390, `com.ibm.jndi.LDAPCtxFactory` is also supported and provides access to an LDAP server. However, this is incompatible with `com.sun.jndi.ldap.LdapCtxFactory`, in that objects created using one InitialContextFactory cannot be read or modified using the other.

You can also use an `InitialContextFactory` that is not in the list above. See “Using an unlisted InitialContextFactory” on page 43 for more details.

**PROVIDER_URL**

The URL of the session’s initial context; the root of all JNDI operations carried out by the tool. Three forms of this property are supported:

- `ldap://hostname/contextname` (for LDAP)
- `file://drive:/pathname` (for file system context)
- `iiop://hostname[:port] /[/?TargetContext=ctx]` (to access base WebSphere Application Server CosNaming namespace)

**SECURITY_AUTHENTICATION**

Whether JNDI passes security credentials to your service provider. This property is used only when an LDAP service provider is used. This property can take one of three values:

- `none` (anonymous authentication)
- `simple` (simple authentication)
- `CRAM-MD5` (CRAM-MD5 authentication mechanism)

If a valid value is not supplied, the property defaults to `none`. See “Security” on page 43 for more details about security with the administration tool.

These properties are set in a configuration file. When you invoke the tool, you can specify this configuration by using the `-cfg` command-line parameter, as described
Using an unlisted InitialContextFactory

You can use the administration tool to connect to JNDI contexts other than those listed in Configuration on page 42 by using three parameters defined in the JMSAdmin configuration file.

To use a different InitialContextFactory:
1. Set the INITIAL_CONTEXT_FACTORY property to the required class name.
2. Define the behavior of the InitialContextFactory using the USE_INITIAL_DIR_CONTEXT, NAME_PREFIX and NAME_READABILITY_MARKER properties.

The settings for these properties are described in the sample configuration file comments.

You do not need to define the three properties listed here, if you use one of the supported INITIAL_CONTEXT_FACTORY values. However, you can give them values to override the system defaults. If you omit one or more of the three InitialContextFactory properties, the administration tool provides suitable defaults based on the values of the other properties.

Security

You need to understand the effect of the SECURITY_AUTHENTICATION property described in Configuration on page 42:

- If you set this parameter to none, JNDI does not pass any security credentials to the service provider, and anonymous authentication is performed.
- If you set the parameter to either simple or CRAM-MD5, security credentials are passed through JNDI to the underlying service provider. These security credentials are in the form of a user distinguished name (User DN) and password.

If security credentials are required, you are prompted for these when the tool initializes. Avoid this by setting the PROVIDER_USERDN and PROVIDER_PASSWORD properties in the JMSAdmin configuration file.
Configuration

**Note:** If you do not use these properties, the text typed, including the password, is echoed to the screen. This may have security implications.

The tool does no authentication itself; the task is delegated to the LDAP server. The LDAP server administrator must set up and maintain access privileges to different parts of the directory. If authentication fails, the tool displays an appropriate error message and terminates.

More detailed information about security and JNDI is in the documentation at Sun’s Java web site (http://java.sun.com).

**Configuring for WebSphere Application Server V3.5**

For the administration tool (or any client application that needs to do subsequent lookups) to work with WebSphere Application Server’s CosNaming repository, you need the following configuration:

- CLASSPATH must include WebSphere Application Server’s JNDI-related jar file, `<WSAppserver>\lib\ujc.jar`
- PATH must include `<WSAppserver>\jdk\jre\bin`, where `<WSAppserver>` is the install path for WebSphere Application Server

**Administration commands**

When the command prompt is displayed, the tool is ready to accept commands. Administration commands are generally of the following form:

`verb [param]*`

where `verb` is one of the administration verbs listed in [Table 7](#). All valid commands consist of at least one (and only one) verb, which appears at the beginning of the command in either its standard or short form.

The parameters a verb can take depend on the verb. For example, the END verb cannot take any parameters, but the DEFINE verb can take any number of parameters. Details of the verbs that take at least one parameter are discussed in later sections of this chapter.

**Table 7. Administration verbs**

<table>
<thead>
<tr>
<th>Verb</th>
<th>Short form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER</td>
<td>ALT</td>
<td>Change at least one of the properties of a given administered object</td>
</tr>
<tr>
<td>DEFINE</td>
<td>DEF</td>
<td>Create and store an administered object, or create a new subcontext</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>DIS</td>
<td>Display the properties of one or more stored administered objects, or the contents of the current context</td>
</tr>
<tr>
<td>DELETE</td>
<td>DEL</td>
<td>Remove one or more administered objects from the namespace, or remove an empty subcontext</td>
</tr>
<tr>
<td>CHANGE</td>
<td>CHG</td>
<td>Alter the current context, allowing the user to traverse the directory namespace anywhere below the initial context (pending security clearance)</td>
</tr>
<tr>
<td>COPY</td>
<td>CP</td>
<td>Make a copy of a stored administered object, storing it under an alternative name</td>
</tr>
<tr>
<td>MOVE</td>
<td>MV</td>
<td>Alter the name under which an administered object is stored</td>
</tr>
</tbody>
</table>
Verb names are not case-sensitive.

Usually, to terminate commands, you press the carriage return key. However, you can override this by typing the + symbol directly before the carriage return. This enables you to enter multiline commands, as shown in the following example:

```
DEFINE Q(BookingsInputQueue) +
  QMGR(QM.POLARIS.TEST) +
  QUEUE(Bookings.Input.Queue) +
  PORT(1415) +
  CCSID(437)
```

Lines beginning with one of the characters *, #, or / are treated as comments, or lines that are ignored.

### Manipulating subcontexts

Use the verbs CHANGE, DEFINE, DISPLAY and DELETE to manipulate directory namespace subcontexts. Their use is described in Table 8.

<table>
<thead>
<tr>
<th>Command syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFINE CTX(ctxName)</td>
<td>Attempts to create a new child subcontext of the current context, having the name ctxName. Fails if there is a security violation, if the subcontext already exists, or if the name supplied is not valid.</td>
</tr>
<tr>
<td>DISPLAY CTX</td>
<td>Displays the contents of the current context. Administered objects are annotated with a, subcontexts with [B]. The Java type of each object is also displayed.</td>
</tr>
<tr>
<td>DELETE CTX(ctxName)</td>
<td>Attempts to delete the current context’s child context having the name ctxName. Fails if the context is not found, is non-empty, or if there is a security violation.</td>
</tr>
</tbody>
</table>
| CHANGE CTX(ctxName)    | Alters the current context, so that it now refers to the child context having the name ctxName. One of two special values of ctxName can be supplied:  
                          =UP moves to the current context’s parent  
                          =INIT moves directly to the initial context  
                          Fails if the specified context does not exist, or if there is a security violation. |

### Administering JMS objects

This section describes the eight types of object that the administration tool can handle. It includes details about each of their configurable properties and the verbs that can manipulate them.

#### Object types

Table 9 on page 46 shows the eight types of administered objects. The Keyword column shows the strings that you can substitute for TYPE in the commands shown...
## Administering JMS objects

As described in Table 10 on page 47.

### Table 9. The JMS object types that are handled by the administration tool

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQConnectionFactory(^1)</td>
<td>CF</td>
<td>The WebSphere MQ implementation of the JMS ConnectionFactory interface. This represents a factory object for creating connections in both the point-to-point and publish/subscribe domains.</td>
</tr>
<tr>
<td>MQQueueConnectionFactory</td>
<td>QCF</td>
<td>The WebSphere MQ implementation of the JMS QueueConnectionFactory interface. This represents a factory object for creating connections in the point-to-point domain.</td>
</tr>
<tr>
<td>MQTTopicConnectionFactory</td>
<td>TCF</td>
<td>The WebSphere MQ implementation of the JMS TopicConnectionFactory interface. This represents a factory object for creating connections in the publish/subscribe domain.</td>
</tr>
<tr>
<td>MQQueue</td>
<td>Q</td>
<td>The WebSphere MQ implementation of the JMS Queue interface. This represents a destination for messages in the point-to-point domain.</td>
</tr>
<tr>
<td>MQTTopic</td>
<td>T</td>
<td>The WebSphere MQ implementation of the JMS Topic interface. This represents a destination for messages in the publish/subscribe domain.</td>
</tr>
<tr>
<td>MQXACConnectionFactory(^2)</td>
<td>XACF</td>
<td>The WebSphere MQ implementation of the JMS XACConnectionFactory interface. This represents a factory object for creating connections in both the point-to-point and publish/subscribe domains, and where the connections use the XA versions of JMS classes.</td>
</tr>
<tr>
<td>MQXAQueueConnectionFactory(^2)</td>
<td>XAQCFC</td>
<td>The WebSphere MQ implementation of the JMS XAQueueConnectionFactory interface. This represents a factory object for creating connections in the point-to-point domain that use the XA versions of JMS classes.</td>
</tr>
<tr>
<td>MQXATopicConnectionFactory(^2)</td>
<td>XATCF</td>
<td>The WebSphere MQ implementation of the JMS XATopicConnectionFactory interface. This represents a factory object for creating connections in the publish/subscribe domain that use the XA versions of JMS classes.</td>
</tr>
</tbody>
</table>
Table 9. The JMS object types that are handled by the administration tool (continued)

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMSWrapXAQueueConnectionFactory&lt;sup&gt;3&lt;/sup&gt;</td>
<td>WSQCF</td>
<td>The WebSphere MQ implementation of the JMS QueueConnectionFactory interface. This represents a factory object for creating connections in the point-to-point domain that use the XA versions of the JMS classes with a version of WebSphere Application Server before Version 5.</td>
</tr>
<tr>
<td>JMSWrapXATopicConnectionFactory&lt;sup&gt;3&lt;/sup&gt;</td>
<td>WSTCF</td>
<td>The WebSphere MQ implementation of the JMS TopicConnectionFactory interface. This represents a factory object for creating connections in the publish/subscribe domain that use the XA versions of the JMS classes with a version of WebSphere Application Server before Version 5.</td>
</tr>
</tbody>
</table>

1. This object type applies to JMS 1.1 only.
2. These classes are provided for use by vendors of application servers. They are unlikely to be directly useful to application programmers.
3. Use this style of ConnectionFactory if you want your JMS sessions to participate in global transactions that are coordinated by a version of WebSphere Application Server before Version 5.

Verbs used with JMS objects

You can use the verbs ALTER, DEFINE, DISPLAY, DELETE, COPY, and MOVE to manipulate administered objects in the directory namespace. Table 10 summarizes their use. Substitute TYPE with the keyword that represents the required administered object, as listed in Table 9 on page 46.

Table 10. Syntax and description of commands used to manipulate administered objects

<table>
<thead>
<tr>
<th>Command syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER TYPE(name) [property]*</td>
<td>Attempts to update the given administered object’s properties with the ones supplied. Fails if there is a security violation, if the specified object cannot be found, or if the new properties supplied are not valid.</td>
</tr>
<tr>
<td>DEFINE TYPE(name) [property]*</td>
<td>Attempts to create an administered object of type TYPE with the supplied properties, and store it under the name name in the current context. Fails if there is a security violation, if the supplied name is not valid or already exists, or if the properties supplied are not valid.</td>
</tr>
<tr>
<td>DISPLAY TYPE(name)</td>
<td>Displays the properties of the administered object of type TYPE, bound under the name name in the current context. Fails if the object does not exist, or if there is a security violation.</td>
</tr>
<tr>
<td>DELETE TYPE(name)</td>
<td>Attempts to remove the administered object of type TYPE, having the name name, from the current context. Fails if the object does not exist, or if there is a security violation.</td>
</tr>
</tbody>
</table>
Creating objects

Objects are created and stored in a JNDI namespace using the following command syntax:

`DEFINE TYPE(name) [property]`*

That is, the `DEFINE` verb, followed by a `TYPE(name)` administered object reference, followed by zero or more `properties` (see “Properties” on page 49).

**LDAP naming considerations**

To store your objects in an LDAP environment, you must give them names that comply with certain conventions. One of these is that object and subcontext names must include a prefix, such as `cn=` (common name), or `ou=` (organizational unit).

The administration tool simplifies the use of LDAP service providers by allowing you to refer to object and context names without a prefix. If you do not supply a prefix, the tool automatically adds a default prefix to the name you supply. For LDAP this is `cn=`.

You can change the default prefix by setting the `NAME_PREFIX` property in the `JMSAdmin` configuration file, as described in “Using an unlisted InitialContextFactory” on page 43.

This is shown in the following example.

```
InitCtx> DEFINE Q(testQueue)

InitCtx> DISPLAY Q(testQueue)
```

You can change the default prefix by setting the `NAME_PREFIX` property in the `JMSAdmin` configuration file, as described in “Using an unlisted InitialContextFactory” on page 43.

This is shown in the following example.

```
InitCtx> DEFINE Q(testQueue)

InitCtx> DISPLAY Q(testQueue)
```

Note that, although the object name supplied (testQueue) does not have a prefix, the tool automatically adds one to ensure compliance with the LDAP naming convention. Likewise, submitting the command `DISPLAY Q(testQueue)` also causes this prefix to be added.
You might need to configure your LDAP server to store Java objects. Information to assist with this configuration is provided in Appendix C, “LDAP schema definition for storing Java objects,” on page 463.

Properties
A property consists of a name-value pair in the format:

```
PROPERTY_NAME(property_value)
```

Property names are not case-sensitive, and are restricted to the set of recognized names shown in Table 11. This table also shows the valid property values for each property.

Table 11. Property names and valid values

<table>
<thead>
<tr>
<th>Property</th>
<th>Short form</th>
<th>Valid values (defaults in bold)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BROKERCCDSUBQ</td>
<td>CCDSUB</td>
<td>SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Any string</td>
</tr>
<tr>
<td>BROKERCCSUBQ</td>
<td>CCSUB</td>
<td>SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Any string</td>
</tr>
<tr>
<td>BROKERCONQ</td>
<td>BCON</td>
<td>Any string</td>
</tr>
<tr>
<td>BROKERDURSUBQ</td>
<td>BDSUB</td>
<td>SYSTEM.JMS.D.SUBSCRIBER.QUEUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Any string</td>
</tr>
<tr>
<td>BROKERPUBQ</td>
<td>BPUB</td>
<td>SYSTEM.BROKER.DEFAULT.STREAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Any string</td>
</tr>
<tr>
<td>BROKERQMGRR</td>
<td>BQM</td>
<td>Any string</td>
</tr>
<tr>
<td>BROKERSUBQ</td>
<td>BSUB</td>
<td>SYSTEM.JMS.ND.SUBSCRIBER.QUEUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Any string</td>
</tr>
<tr>
<td>BROKERVER</td>
<td>BVER</td>
<td>V1 - To use the WebSphere MQ broker. Also to use the WebSphere MQ Integrator V2 or WebSphere MQ Event Broker brokers in compatibility mode. V2 - To use the WebSphere MQ Integrator V2 or WebSphere MQ Event Broker brokers in native mode</td>
</tr>
<tr>
<td>CCSID</td>
<td>CCS</td>
<td>Any positive integer</td>
</tr>
<tr>
<td>CHANNEL</td>
<td>CHAN</td>
<td>Any string</td>
</tr>
<tr>
<td>CLEANUP</td>
<td>CL</td>
<td>SAFE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASPROP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STRONG</td>
</tr>
<tr>
<td>CLEANUPINT</td>
<td>CLINT</td>
<td>3600000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Any positive integer</td>
</tr>
<tr>
<td>CLIENTID</td>
<td>CID</td>
<td>Any string</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>DESC</td>
<td>Any string</td>
</tr>
<tr>
<td>DIRECTAUTH</td>
<td>DAUTH</td>
<td>BASIC - No authentication, username authentication, or password authentication</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CERTIFICATE - Public key certificate authentication</td>
</tr>
<tr>
<td>ENCODING</td>
<td>ENC</td>
<td>See “The ENCODING property” on page 57</td>
</tr>
</tbody>
</table>

Administering JMS objects

Chapter 5. Using the WebSphere MQ JMS administration tool
### Table 11. Property names and valid values (continued)

<table>
<thead>
<tr>
<th>Property</th>
<th>Short form</th>
<th>Valid values (defaults in bold)</th>
</tr>
</thead>
</table>
| EXPIRY           | EXP        | • APP  - Expiry may be defined by the JMS application.  
|                  |            | • UNLIM  - No expiry occurs.  
|                  |            | • Any positive integer representing expiry in milliseconds.                                                                |
| FAILIFQUIESCE    | FIQ        | • Yes  - Applications return from a method call if the queue manager has entered a controlled shutdown.  
|                  |            | • No   - Applications continue to carry out operations against a quiescing queue manager, preventing that queue manager’s shutdown. |
| HOSTNAME         | HOST       | • localhost  
|                  |            | • Any string                                                                                                               |
| LOCALADDRESS     | LA         | • Not set  
|                  |            | • A string in the format: [ip-addr][low-port[high-port]]  
|                  |            | Here are some examples:  
|                  |            | 9.20.4.98 locally  
|                  |            | The channel binds to address 9.20.4.98 locally  
|                  |            | 9.20.4.98(1000) locally and uses port 1000  
|                  |            | The channel binds to address 9.20.4.98 locally and uses a port in the range 1000 to 2000  
|                  |            | (1000) The channel binds to port 1000 locally  
|                  |            | (1000,2000) The channel binds to a port in the range 1000 to 2000 locally  
|                  |            | You can specify a host name instead of an IP address.  
|                  |            | For direct connections, this property applies only when multicast is used and the value of the property must not contain a port number. If it does contain a port number, the connection is rejected.  
|                  |            | Therefore, the only valid values of the property are null, an IP address, or a host name.                                    |
| MSGBATCHSZ       | MBS        | • 10  
|                  |            | • Any positive integer                                                                                                         |
| MSGRETENTION     | MRET       | • Yes  - Unwanted messages remain on the input queue.  
|                  |            | • No   - Unwanted messages are dealt with according to their disposition options                                                |
| MSGSELECTION     | MSEL       | • CLIENT  - Message selection is done by the client.  
|                  |            | • BROKER - Message selection is done by the broker.                                                                                |
### Table 11. Property names and valid values (continued)

<table>
<thead>
<tr>
<th>Property</th>
<th>Short form</th>
<th>Valid values (defaults in bold)</th>
</tr>
</thead>
</table>
| MULTICAST     | MCAST      | • **DISABLED** - Multicast is disabled. This is the default value for ConnectionFactory and TopicConnectionFactory objects.  
• **ASCF** - Same as the setting for the ConnectionFactory or TopicConnectionFactory object. This value is valid only for Topic objects, and is the default value for Topic objects.  
• **RELIABLE** - Multicast is enabled with reliable delivery only.  
• **ENABLED** - Multicast is enabled if it is available. Using this value might provide a reliable multicast connection depending on the server configuration.  
• **NOTR** - As **ENABLED**, but does not provide a reliable multicast connection. This value is used to enable multicast for legacy applications. |
| PERSISTENCE   | PER        | • **APP** - Persistence is defined by the JMS application.  
• **QDEF** - Persistence takes the value of the queue default.  
• **PERS** - Messages are persistent.  
• **NON** - Messages are non-persistent. |
| POLLINGINT    | PINT       | • **5000**  
• Any positive integer |
| PORT          |            | • **1414** (for **TRANSPORT** set to **BIND** or **CLIENT**); **1506** (for **TRANSPORT** set to **DIRECT**)  
• Any positive integer |
| PRIORITY      | PRI        | • **APP** - Priority is defined by the JMS application.  
• **QDEF** - Priority takes the value of the queue default.  
• Any integer in the range 0-9. |
| PROXYHOSTNAME | PHOST      | • **Not set**  
• The host name of the proxy server |
| PROXYPORT     | PPORT      | • **443**  
• The port number of the proxy server |
| PUBACKINT     | PAI        | • **25**  
• Any positive integer |
| QMANAGER      | QMGR       | Any string |
| QUEUE         | QU         | Any string |
| RECEXIT       | RCX        | Any string |
| RECEXITINIT   | RCXI       | Any string |
| SECEXIT       | SCX        | Any string |
| SECEXITINIT   | SCXI       | Any string |
| SENDEXIT      | SDXI       | Any string |
| SENDXITINIT   | SDXI       | Any string |
| SPARSESUBS    | SSUBS      | • **NO** - Subscriptions receive frequent matching messages.  
• **YES** - Subscriptions receive infrequent matching messages. This value requires that the subscription queue can be opened for browse. |
### Property names and valid values (continued)

<table>
<thead>
<tr>
<th>Property</th>
<th>Short form</th>
<th>Valid values (defaults in bold)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSLCIPHERSUITE</td>
<td>SCPHS</td>
<td>• Not set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• See “SSL properties” on page 58</td>
</tr>
<tr>
<td>SSLCRL</td>
<td>SCRL</td>
<td>• Not set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Space-separated list of LDAP URLs. See “SSL properties” on page 58</td>
</tr>
<tr>
<td>SSLPEERNAME</td>
<td>SPEER</td>
<td>• Not set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• See “SSL properties” on page 58</td>
</tr>
<tr>
<td>STATREFRESHINT</td>
<td>SRI</td>
<td>• 60000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Any positive integer</td>
</tr>
<tr>
<td>SUBSTORE</td>
<td>SS</td>
<td>• MIGRATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• QUEUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BROKER</td>
</tr>
<tr>
<td>SYNCPOINTALLGETS</td>
<td>SPAG</td>
<td>• No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Yes</td>
</tr>
<tr>
<td>TARGCLIENT</td>
<td>TC</td>
<td>• JMS - The target of the message is a JMS application.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MQ - The target of the message is a non-JMS WebSphere MQ application.</td>
</tr>
<tr>
<td>TEMPMODEL</td>
<td>TM</td>
<td>Any string</td>
</tr>
<tr>
<td>TEMPPREFIX</td>
<td>TQP</td>
<td>Any string</td>
</tr>
<tr>
<td>TOPIC</td>
<td>TOP</td>
<td>Any string</td>
</tr>
<tr>
<td>TRANSPORT</td>
<td>TRAN</td>
<td>• BIND - For a bindings connection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CLIENT - For a client connection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DIRECT - For a direct connection to a WebSphere MQ Event Broker, WebSphere Business Integration Event Broker, or WebSphere Business Integration Message Broker broker</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DIRECTHTTP - For a direct connection using HTTP tunnelling.</td>
</tr>
<tr>
<td>USECONNPOOLING</td>
<td>UCP</td>
<td>• Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• No</td>
</tr>
</tbody>
</table>

**Notes:**

1. In certain environments, specifying the same queue name for the BROKERCCDSUBQ and BROKERDURSUBQ properties of an MQTopic object can cause a JMSException to be thrown. You are advised, therefore, to specify different queue names for these properties.

Many of the properties are relevant only to a specific subset of the object types. Table 12 on page 53 shows for each property which object types are valid, and gives a brief description of each property. The object types are identified using keywords; refer to Table 9 on page 46 for an explanation of these.

Numbers refer to notes at the end of the table. See also “Property dependencies” on page 56. Appendix A, “Mapping between administration tool properties and programmable properties,” on page 457 shows the relationship between properties set by the tool and programmable properties.
<table>
<thead>
<tr>
<th>Property</th>
<th>CI</th>
<th>QCF</th>
<th>TCF</th>
<th>Q</th>
<th>T</th>
<th>XACF</th>
<th>WSQCF</th>
<th>XAQCF</th>
<th>WSTCF</th>
<th>XATCF</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BROKERCCDSUBQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The name of the queue from which durable subscription messages are retrieved for a ConnectionConsumer</td>
</tr>
<tr>
<td>BROKERCCSUBQ</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The name of the queue from which non-durable subscription messages are retrieved for a ConnectionConsumer</td>
</tr>
<tr>
<td>BROKERCONQ</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Broker’s control queue name</td>
</tr>
<tr>
<td>BROKERDURSUBQ</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The name of the queue from which non-durable subscription messages are retrieved</td>
</tr>
<tr>
<td>BROKERHOSTNAME</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The name of the queue from which non-durable subscription messages are retrieved</td>
</tr>
<tr>
<td>BROKERQMgr</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The queue manager on which the broker is running</td>
</tr>
<tr>
<td>BROKERQSUBQ</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The name of the queue from which non-durable subscription messages are retrieved</td>
</tr>
<tr>
<td>BROKERSERVER</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The version of the broker being used</td>
</tr>
<tr>
<td>CCSID</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The coded-character-set-ID to be used on connections</td>
</tr>
<tr>
<td>CHANNEL</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The name of the client connection channel being used</td>
</tr>
<tr>
<td>CLEANUP</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Cleanup Level for BROKER or MIGRATE Subscription Stores</td>
</tr>
<tr>
<td>CLEANUPINT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The interval between background executions of the publish/subscribe cleanup utility</td>
</tr>
<tr>
<td>CLIENTID</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A string identifier for the client</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A description of the stored object</td>
</tr>
<tr>
<td>DIRECTAUTH</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>To enable SSL authentication for a direct connection</td>
</tr>
<tr>
<td>ENCODING</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The encoding scheme used for this destination</td>
</tr>
<tr>
<td>EXPIRY</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The period after which messages at a destination expire</td>
</tr>
<tr>
<td>HOSTNAME</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The name of the host on which the queue manager or WebSphere MQ Event Broker broker resides. A dotted-decimal TCP/IP address can also be used.</td>
</tr>
<tr>
<td>LOCALADDRESS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The range of local ports to be used when making a connection to a WebSphere MQ queue manager</td>
</tr>
<tr>
<td>MSGSBSZ</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The maximum number of messages to be taken from a queue in one packet when using asynchronous message delivery</td>
</tr>
</tbody>
</table>

Table 12. The valid combinations of property and object type
### Table 12. The valid combinations of property and object type (continued)

<table>
<thead>
<tr>
<th>Property</th>
<th>CF</th>
<th>QCF</th>
<th>TCF</th>
<th>QT</th>
<th>XACF</th>
<th>WSQCF</th>
<th>XAQCFC</th>
<th>WSTCF</th>
<th>XATCF</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSGRETENTION</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Whether or not the connection consumer keeps unwanted messages on the input queue</td>
</tr>
<tr>
<td>MSGSELECTION</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Determines whether message selection is done by the JMS client or by the broker. If TRANSPORT has the value DIRECT, message selection is always done by the broker and the value of MSGSELECTION is ignored. Message selection by the broker is not supported when BROKERVER has the value V1.</td>
</tr>
<tr>
<td>MULTICAST</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>To enable multicast on a direct connection</td>
</tr>
<tr>
<td>PERSISTENCE</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The persistence of messages sent to a destination</td>
</tr>
<tr>
<td>POLLINGINT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The interval, in milliseconds, between scans of all receivers during asynchronous message delivery</td>
</tr>
<tr>
<td>PORT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The port on which the queue manager or broker listens</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The priority for messages sent to a destination</td>
</tr>
<tr>
<td>PROXYHOSTNAME</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The host name of the proxy server for a direct connection</td>
</tr>
<tr>
<td>PROXYPORF</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The port number of the proxy server for a direct connection</td>
</tr>
<tr>
<td>PUBACKINT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The interval, in number of messages, between publish requests that require acknowledgement from the broker</td>
</tr>
<tr>
<td>QMANAGER</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td>The name of the queue manager to connect to</td>
</tr>
<tr>
<td>QUEUE</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The underlying name of the queue representing this destination</td>
</tr>
<tr>
<td>RECEEXIT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The fully-qualified class name of the receive exit being used</td>
</tr>
<tr>
<td>RECEEXITINIT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The receive exit initialization string</td>
</tr>
<tr>
<td>SECEXIT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The fully-qualified class name of the security exit being used</td>
</tr>
<tr>
<td>SECEEXITINIT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The security exit initialization string</td>
</tr>
<tr>
<td>SENDEXIT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The fully-qualified class name of the send exit being used</td>
</tr>
<tr>
<td>SENDEXITINIT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The send exit initialization string</td>
</tr>
<tr>
<td>SPARSESUBS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Controls the message retrieval policy of a TopicSubscriber object</td>
</tr>
<tr>
<td>SSLCIPHERSUITE</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The cipher suite to use for SSL connection</td>
</tr>
</tbody>
</table>
Table 12. The valid combinations of property and object type (continued)

<table>
<thead>
<tr>
<th>Property</th>
<th>CF1</th>
<th>QCF</th>
<th>TCF</th>
<th>Q</th>
<th>T</th>
<th>XACF</th>
<th>WSQCF</th>
<th>XATCF</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSLCRL</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CRL servers to check for SSL certificate revocation</td>
</tr>
<tr>
<td>SSLPEERNAME</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>For SSL, a distinguished name skeleton that must match that provided by the queue manager</td>
</tr>
<tr>
<td>STATREFRESHINT</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The interval, in milliseconds, between transactions to refresh publish/subscribe status</td>
</tr>
<tr>
<td>SUBSTORE</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Where WebSphere MQ JMS should store persistent data relating to active subscriptions</td>
</tr>
<tr>
<td>SYNCPOINTALLGETS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td>Whether all gets should be performed under syncpoint</td>
</tr>
<tr>
<td>TARGCLIENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td>Whether the WebSphere MQ RFH2 format is used to exchange information with target applications</td>
</tr>
<tr>
<td>TEMPMODEL</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The name of the model queue from which temporary queues are created</td>
</tr>
<tr>
<td>TEMPPREFIX</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The prefix that is used to form the name of a WebSphere MQ dynamic queue. The rules for forming the prefix are the same as those for forming the contents of the DynamicQName field in a WebSphere MQ object descriptor, structure MQOD, but the last non blank character must be an asterisk. If no value is specified for the property, the value used is CSQ.* on z/OS and AMQ.* on the other platforms.</td>
</tr>
<tr>
<td>TOPIC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td>The underlying name of the topic representing this destination</td>
</tr>
<tr>
<td>TRANSPORT</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Whether connections use the WebSphere MQ bindings, a client connection, or WebSphere MQ Event Broker.</td>
</tr>
<tr>
<td>USECONNPOOLING</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td>Whether to use connection pooling</td>
</tr>
</tbody>
</table>
Table 12. The valid combinations of property and object type (continued)

<table>
<thead>
<tr>
<th>Property</th>
<th>Cl</th>
<th>QCF</th>
<th>TCF</th>
<th>Q</th>
<th>T</th>
<th>XACF</th>
<th>WSQCF</th>
<th>WSTCF</th>
<th>Description</th>
</tr>
</thead>
</table>

Notes:

1. This object type applies to JMS 1.1 only.
2. Only the BROKERVER, CLIENTID, DESCRIPTION, HOSTNAME, PORT, and TRANSPORT properties are supported for a TopicConnectionFactory object, or a JMS 1.1 domain independent ConnectionFactory object, when connecting directly to WebSphere MQ Event Broker over TCP/IP.
3. See Appendix D, “Connecting to other products,” on page 469.
4. HOSTNAME, PORT, and TRANSPORT are also used to identify if you are connecting to WebSphere MQ Event Broker and the broker’s IP hostname and listening port. For more information about using WebSphere MQ Event Broker, see Chapter 11, “Writing WebSphere MQ JMS publish/subscribe applications,” on page 213.
5. The TARGCLIENT property indicates whether the WebSphere MQ RFH2 format is used to exchange information with target applications.
   - The MQJMS_CLIENT_JMS_COMPLIANT constant indicates that the RFH2 format is used to send information. Applications that use WebSphere MQ JMS understand the RFH2 format. Set the MQJMS_CLIENT_JMS_COMPLIANT constant when you exchange information with a target WebSphere MQ JMS application.
   - The MQJMS_CLIENT_NONJMS_MQ constant indicates that the RFH2 format is not used to send information. Typically, this value is used for an existing WebSphere MQ application (that is, one that does not handle RFH2).
6. For XACF, XAQCF, XATCF, WSQCF, and WSTCF objects, only the BIND transport type is allowed.

Property dependencies

Some properties have dependencies on each other. This might mean that it is meaningless to supply a property unless another property is set to a particular value. The specific property groups where this can occur are:

- Client properties
- Properties for connecting to WebSphere MQ Event Broker
- Exit initialization strings

Client properties

Some properties are only relevant to a connection with the TRANSPORT property set to the value CLIENT. If this property is not explicitly set on a connection factory to one of the values CLIENT or DIRECT, the transport used on connections provided by the factory is WebSphere MQ Bindings. Consequently, none of the client properties on this connection factory can be configured. These are:

- HOST
- PORT
- CHANNEL
- CCSID
- RECEXIT
- RECEXITINIT
- SECEXIT
- SECEXITINIT
- SENDEXIT
- SENDEXITINIT
- SSLCIPHERSUITENe
• SSLCRL
• SSLPEERNAME

It is an error to set any of these properties without setting the TRANSPORT property to CLIENT (or, for some, DIRECT; see “Properties for connecting to WebSphere MQ Event Broker”).

Properties for connecting to WebSphere MQ Event Broker
The only properties used with a direct connection to WebSphere MQ Event Broker are BROKERVER, CLIENTID, DESCRIPTION, HOSTNAME, PORT, and TRANSPORT.

The default values for PORT and BROKERVER are set by the definition of TRANSPORT:
1. Defining a connection factory with TRANSPORT as CLIENT sets:
   • BROKERVER to V1
   • PORT to 1414
2. Defining a connection factory with TRANSPORT as DIRECT sets:
   • BROKERVER to V2
   • PORT to 1506

If you explicitly set the value of PORT or BROKERVER, a later change to the value of TRANSPORT does not override your choices.

Exit initialization strings
Do not set any of the exit initialization strings without supplying the corresponding exit name. The exit initialization properties are:
• RECEXITINIT
• SECEXITINIT
• SENDEXITINIT

For example, specifying RECEXITINIT(myString) without specifying RECEXIT(some.exit.classname) causes an error.

The ENCODING property
The valid values that the ENCODING property can take are constructed from three sub-properties:

integer encoding Either normal or reversed
decimal encoding Either normal or reversed
floating-point encoding IEEE normal, IEEE reversed, or z/OS.

The ENCODING is expressed as a three-character string with the following syntax:
{N|R}{N|R}{N|R|3}

In this string:
• N denotes normal
• R denotes reversed
• 3 denotes z/OS
• The first character represents integer encoding
• The second character represents decimal encoding
• The third character represents floating-point encoding

This provides a set of twelve possible values for the ENCODING property.
There is an additional value, the string `NATIVE`, which sets appropriate encoding values for the Java platform.

The following examples show valid combinations for `ENCODING`:

- `ENCODING(NR)`
- `ENCODING(NATIVE)`
- `ENCODING(RR3)`

### SSL properties

When you specify `TRANSPORT(CLIENT)`, you can enable Secure Sockets Layer (SSL) encrypted communication using the `SSLCIPHERSUITE` property. Set this property to a valid CipherSuite provided by your JSSE provider; it must match the `CipherSpec` named on the `SVRCONN` channel named by the `CHANNEL` property.

However, `CipherSpecs` (as specified on the `SVRCONN` channel) and `CipherSuites` (as specified on `ConnectionFactory` objects) use different naming schemes to represent the same SSL encryption algorithms. If a recognized `CipherSpec` name is specified on the `SSLCIPHERSUITE` property, JMSAdmin issues a warning and maps the `CipherSpec` to its equivalent `CipherSuite`. See Appendix H, “SSL CipherSuites supported by WebSphere MQ,” on page 487 for a list of `CipherSpecs` recognized by WebSphere MQ and JMSAdmin.

The `SSLPEERNAME` matches the format of the `SSLPEER` parameter, which can be set on channel definitions. It is a list of attribute name and value pairs separated by commas or semicolons. For example:

```
SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

The set of names and values makes up a *distinguished name*. For more details about distinguished names and their use with WebSphere MQ, see the *WebSphere MQ Security* book.

The example given checks the identifying certificate presented by the server at connect-time. For the connection to succeed, the certificate must have a Common Name beginning QMGR., and must have at least two Organizational Unit names, the first of which is IBM and the second WEBSPHERE. Checking is case-insensitive.

If `SSLPEERNAME` is not set, no such checking is performed. `SSLPEERNAME` is ignored if `SSLCIPHERSUITE` is not specified.

The `SSLCRL` property specifies zero or more CRL (Certificate Revocation List) servers. Use of this property requires a JVM at Java 2 v1.4. This is a space-delimited list of entries of the form:

```
ldap://hostname:[port]
```

optionally followed by a single `/`. If `port` is omitted, the default LDAP port of 389 is assumed. At connect-time, the SSL certificate presented by the server is checked against the specified CRL servers. See the *WebSphere MQ Security* book for more about CRL security.

If `SSLCRL` is not set, no such checking is performed. `SSLCRL` is ignored if `SSLCIPHERSUITE` is not specified.
Sample error conditions

The following are examples of the error conditions that might arise when creating an object:

CipherSpec mapped to CipherSuite

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) SSLCIPHERSUITE(RC4_MD5_US)
WARNING: Converting CipherSpec RC4_MD5_US to CipherSuite SSL_RSA_WITH_RC4_128_MD5
```

Invalid property for object

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PRIORITY(4)
Unable to create a valid object, please check the parameters supplied
Invalid property for a QCF: PRI
```

Invalid type for property value

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) CCSID(english)
Unable to create a valid object, please check the parameters supplied
Invalid value for CCS property: English
```

Property clash - client/bindings

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) HOSTNAME(polaris.hursley.ibm.com)
Unable to create a valid object, please check the parameters supplied
Invalid property in this context: Client-bindings attribute clash
```

Property clash - Exit initialization

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) SECEXITINIT(initStr)
Unable to create a valid object, please check the parameters supplied
Invalid property in this context: ExitInit string supplied without Exit string
```

Property value outside valid range

```
InitCtx/cn=Trash> DEFINE Q(testQ) PRIORITY(12)
Unable to create a valid object, please check the parameters supplied
Invalid value for PRI property: 12
```

Unknown property

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PIZZA(ham and mushroom)
Unable to create a valid object, please check the parameters supplied
Unknown property: PIZZA
```

The following are examples of error conditions that might arise on Windows when looking up JNDI administered objects from a JMS client. If your JMS application is running in a WebSphere Application Server environment, these error conditions might occur only if you are using a version of WebSphere Application Server before Version 5.

1. If you are using the WebSphere JNDI provider,
   com.ibm.websphere.naming.WsnInitialContextFactory, you must use a forward slash (/) to access administered objects defined in sub-contexts; for example, jms/MyQueueName. If you use a backslash (\), an InvalidNameException is thrown.

2. If you are using the Sun JNDI provider,
   com.sun.jndi.fscontext.RefSCContextFactory, you must use a backslash (\) to access administered objects defined in sub-contexts; for example, ctx1\fred. If you use a forward slash (/), a NameNotFoundException is thrown.
Administering JMS objects
Part 2. Programming with WebSphere MQ base Java

Chapter 6. Introduction for programmers .................................................. 63
Why should I use the Java interface? .................................................... 63
The WebSphere MQ classes for Java interface ..................................... 64
Java Development Kit ........................................................................ 64
WebSphere MQ classes for Java class library ..................................... 65

Chapter 7. Writing WebSphere MQ base Java programs ......................... 67
Should I write applets or applications? ................................................. 67
Connection differences .................................................................... 67
Client connections ............................................................................ 67
Bindings mode .................................................................................. 68
Defining which connection to use ..................................................... 68
Specifying a range of ports for client connections ............................... 68
Example code fragments .................................................................. 69
Example applet code ......................................................................... 69
Example application code ................................................................. 72
Operations on queue managers ......................................................... 74
Setting up the WebSphere MQ environment ...................................... 74
Connecting to a queue manager ......................................................... 75
Accessing queues and processes ....................................................... 75
Handling messages ........................................................................... 76
Handling errors ................................................................................ 77
Getting and setting attribute values ................................................ 78
Multithreaded programs ................................................................... 79
Writing user exits ............................................................................. 79
Connection pooling ........................................................................... 80
Controlling the default connection pool .......................................... 81
The default connection pool and multiple components .................... 83
Supplying a different connection pool .............................................. 84
Supplying your own ConnectionManager ........................................ 85
JTA/JDBC coordination using WebSphere MQ base Java .................. 87
Installation ....................................................................................... 87
Installation on Windows systems ....................................................... 87
Installation on other platforms ........................................................ 87
Usage ............................................................................................... 88
Known problems and limitations ....................................................... 88
Secure Sockets Layer (SSL) support ................................................ 89
Enabling SSL ................................................................................... 90
Using the distinguished name of the queue manager ......................... 90
Using certificate revocation lists ....................................................... 91
Supplying a customized SSLSocketFactory ..................................... 92
Error handling when using SSL ........................................................ 92
Compiling and testing WebSphere MQ base Java programs .............. 93
Running WebSphere MQ base Java applets ...................................... 93
Running WebSphere MQ base Java applications ............................ 94
Tracing WebSphere MQ base Java programs .................................... 94

Chapter 8. Environment-dependent behavior ........................................ 95
Core details ...................................................................................... 95
Restrictions and variations for core classes ...................................... 96
MQGMO_* values ......................................................................... 96
MQPMRF_* values ....................................................................... 96
MQPMO_* values ....................................................................... 96
MQCNO_FASTPATH_BINDING ....................................................... 96
MQRO_* values ....................................................................... 97
Miscellaneous differences with z/OS and OS/390 .............................. 97
Features outside the core .................................................................. 98
MQQueueManager constructor option .............................................. 98
MQQueueManager.begin() method .................................................. 98
MQGetMessageOptions fields .......................................................... 98
Distribution lists ............................................................................ 98
MQPutMessageOptions fields ........................................................... 98
MQMD fields ............................................................................... 99

Chapter 9. The WebSphere MQ base Java classes and interfaces .......... 101
MQChannelDefinition ..................................................................... 102
Variables ....................................................................................... 102
Constructors. ................................................................................ 103
MQChannelExit ............................................................................. 104
Variables ....................................................................................... 104
Constructors. ................................................................................ 106
MQDistributionList ........................................................................ 107
Constructors. ................................................................................ 107
Methods ....................................................................................... 107
MQDistributionListItem .................................................................. 109
Variables ....................................................................................... 109
Constructors. ................................................................................ 109
Methods ....................................................................................... 109
MQException ................................................................................. 117
Variables ....................................................................................... 117
Constructors. ................................................................................ 117
Methods ....................................................................................... 118
MQGetMessageOptions .................................................................. 119
Variables ....................................................................................... 119
Constructors. ................................................................................ 122
Methods ....................................................................................... 123
MQManagedObject .......................................................................... 123
Variables ....................................................................................... 123
Constructors. ................................................................................ 124
Methods ....................................................................................... 124
MQMessage .................................................................................... 126
Variables ....................................................................................... 126
Constructors. ................................................................................ 134
Methods ....................................................................................... 134
MQMessageTracker ................................................................-------- 144
Variables ....................................................................................... 144
MQPoolServices ............................................................................. 146
Constructors. ................................................................................ 146
Methods ....................................................................................... 146
MQPoolServicesEvent ..................................................................... 147
Variables ....................................................................................... 147
Constructors. ................................................................................ 148
Methods ....................................................................................... 148
MQPoolToken ................................................................................ 149
Using Java
Chapter 6. Introduction for programmers

This chapter contains general information for programmers. For more detailed information about writing programs, see Chapter 7, “Writing WebSphere MQ base Java programs,” on page 67.

Why should I use the Java interface?

The WebSphere MQ classes for Java programming interface makes the many benefits of Java available to you as a developer of WebSphere MQ applications:

- The Java programming language is easy to use.
  There is no need for header files, pointers, structures, unions, and operator overloading. Programs written in Java are easier to develop and debug than their C and C++ equivalents.

- Java is object-oriented.
  The object-oriented features of Java are comparable to those of C++, but there is no multiple inheritance. Instead, Java uses the concept of an interface.

- Java is inherently distributed.
  The Java class libraries contain a library of routines for coping with TCP/IP protocols like HTTP and FTP. Java programs can access URLs as easily as accessing a file system.

- Java is robust.
  Java puts a lot of emphasis on early checking for possible problems, dynamic (runtime) checking, and the elimination of situations that are error prone. Java uses a concept of references that eliminates the possibility of overwriting memory and corrupting data.

- Java is secure.
  Java is intended to be run in networked or distributed environments, and a lot of emphasis has been placed on security. Java programs cannot overrun their runtime stack and cannot corrupt memory outside their process space. When Java programs are downloaded from the Internet, they cannot even read or write local files.

- Java programs are portable.
  There are no implementation-dependent aspects of the Java specification. The Java compiler generates an architecture-neutral object file format. The compiled code is executable on many processors, as long as the Java runtime system is present.

If you write your application using WebSphere MQ classes for Java, users can download the Java byte codes (called applets) for your program from the Internet. Users can then run these applets on their own machines. This means that users with access to your Web server can load and run your application with no prior installation needed on their machines.

When an update to the program is required, you update the copy on the Web server. The next time that users access the applet, they automatically receive the latest version. This can significantly reduce the costs involved in installing and updating traditional client applications where a large number of desktops are involved.
Advantages of Java

If you place your applet on a Web server that is accessible outside the corporate firewall, anyone on the Internet can download and use your application. This means that you can get messages into your WebSphere MQ system from anywhere on the Internet. This opens the door to building a whole new set of Internet accessible service, support, and electronic commerce applications.

The WebSphere MQ classes for Java interface

The procedural WebSphere MQ application programming interface is built around the following verbs:

MQBACK, MQBEGIN, MQCLOSE, MQCMIT, MQCONN, MQCONNX,
MQDISC, MQGET, MQINQ, MQOPEN, MQPUT, MQPUT1, MQSET

These verbs all take, as a parameter, a handle to the WebSphere MQ object on which they are to operate. Because Java is object-oriented, the Java programming interface turns this round. Your program consists of a set of WebSphere MQ objects, which you act upon by calling methods on those objects.

When you use the procedural interface, you disconnect from a queue manager by using the call MQDISC(Hconn, CompCode, Reason), where Hconn is a handle to the queue manager.

In the Java interface, the queue manager is represented by an object of class MQQueueManager. You disconnect from the queue manager by calling the disconnect() method on that class.

```java
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
// disconnect from the queue manager
queueManager.disconnect();
```

Java Development Kit

Before you can compile any applets or applications that you write, you must have access to a Java Development Kit (JDK) for your development platform. The JDK contains all the standard Java classes, variables, constructors, and interfaces on which the WebSphere MQ classes for Java classes depend. It also contains the tools required to compile and run the applets and programs on each supported platform.

You can download IBM Developer Kits for Java from the IBM Software Download Catalog, which is available on the World Wide Web at location:

http://www.ibm.com/developerworks/java

To compile Java applications on the iSeries and AS/400 platforms, you must first install:

- The AS/400 Developer Kit for Java, 5769-JV1
- The Qshell Interpreter, OS/400 (5769-SS1) Option 30
WebSphere MQ classes for Java class library

WebSphere MQ classes for Java is a set of Java classes that enable Java applets and applications to interact with WebSphere MQ.

The following classes are provided:
- MQChannelDefinition
- MQChannelExit
- MQDistributionList
- MQDistributionListItem
- MQEnvironment
- MQException
- MQGetMessageOptions
- MQManagedObject
- MQMessage
- MQMessageTracker
- MQPoolServices
- MQPoolServicesEvent
- MQPoolToken
- MQPutMessageOptions
- MQProcess
- MQQueue
- MQQueueManager
- MQSimpleConnectionManager

The following Java interfaces are provided:
- MQC
- MQPoolServicesEventListener
- MQReceiveExit
- MQSecurityExit
- MQSendExit

Implementation of the following Java interfaces is also provided. However, these interfaces are not intended for direct use by applications:
- MQConnectionManager
- javax.resource.spi.ManagedConnection
- javax.resource.spi.ManagedConnectionFactory
- javax.resource.spi.ManagedConnectionMetaData

In Java, a package is a mechanism for grouping sets of related classes together. The WebSphere MQ classes and interfaces are shipped as a Java package called com.ibm.mq. To include the WebSphere MQ classes for Java package in your program, add the following line at the top of your source file:

```java
import com.ibm.mq.*;
```
WebSphere MQ base Java class library
Chapter 7. Writing WebSphere MQ base Java programs

To use WebSphere MQ classes for Java to access WebSphere MQ queues, you write Java programs that contain calls that put messages onto, and get messages from, WebSphere MQ queues. The programs can take the form of Java applets, Java servlets, or Java applications.

This chapter provides information to assist with writing Java applets, servlets, and applications to interact with WebSphere MQ systems. For details of individual classes, see Chapter 9, “The WebSphere MQ base Java classes and interfaces,” on page 101.

Should I write applets or applications?

Whether you write applets, servlets, or applications depends on the connection that you want to use and from where you want to run the programs.

The main differences between applets, servlets, and applications are:

• Applets are run with an applet viewer or in a Web browser, servlets are run in a Web application server, and applications are run standalone.
• Applets can be downloaded from a Web server to a Web browser machine, but applications and servlets are not.
• Applets run with additional security rules limiting what they can do.

See Appendix F, “Using WebSphere MQ Java in applets with Java 1.2 or later,” on page 481 for more information about this.

The following general rules apply:

• If you want to run your programs from machines that do not have WebSphere MQ classes for Java installed locally, write applets.
• The native bindings mode of WebSphere MQ classes for Java does not support applets. Therefore, if you want to use your programs in all connection modes, including the native bindings mode, write servlets or applications.

Connection differences

The way you program for WebSphere MQ classes for Java has some dependencies on the connection modes you want to use.

Client connections

When WebSphere MQ classes for Java is used as a client, it is similar to the WebSphere MQ C client, but has the following differences:

• It supports only TCP/IP.
• It does not support connection tables.
• It does not read any WebSphere MQ environment variables at startup.
• Information that would be stored in a channel definition and in environment variables is stored in a class called Environment. Alternatively, this information can be passed as parameters when the connection is made.
• Error and exception conditions are written to a log specified in the MQException class. The default error destination is the Java console.
Connection differences

The WebSphere MQ classes for Java clients do not support the MQBEGIN verb or fast bindings.

For general information on WebSphere MQ clients, see the WebSphere MQ Clients book.

Bindings mode

The bindings mode of WebSphere MQ classes for Java differs from the client modes in the following ways:

- Most of the parameters provided by the MQEnvironment class are ignored
- The bindings support the MQBEGIN verb and fast bindings into the WebSphere MQ queue manager

Note: WebSphere MQ for iSeries and WebSphere MQ for z/OS do not support the use of MQBEGIN to initiate global units of work that are coordinated by the queue manager.

Defining which connection to use

The connection is determined by the setting of variables in the MQEnvironment class.

MQEnvironment.properties

This can contain the following key/value pairs:

- For client and bindings connections:
  - MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES

MQEnvironment.hostname

Set the value of this variable follows:

- For client connections, set this to the host name of the WebSphere MQ server to which you want to connect
- For bindings mode, set this to null

Specifying a range of ports for client connections

If a JMS application attempts to connect to a WebSphere MQ queue manager in client mode, a firewall might allow only those connections that originate from specified ports or a range of ports. In this situation, you can specify a port, or a range of points, that the application can bind to. You can do this in either of the following ways:

- You can add a key-value pair to the properties variable in the MQEnvironment class. The relevant key is MQC.LOCAL_ADDRESS_PROPERTY. Here is an example:
  
  (MQEnvironment.properties).put(MQC.LOCAL_ADDRESS_PROPERTY, "9.20.0.1(2000,3000)";

- You can set the localAddressSetting variable in the MQEnvironment class. Here is an example:
  
  MQEnvironment.localAddressSetting = "9.20.0.1(2000,3000)";

In each of these examples, when the application connects to a queue manager subsequently, the application binds to a local IP address and port number in the range 9.20.0.1(2000) to 9.20.0.1(3000).

Connection errors might occur if you restrict the range of ports. If an error occurs, an MQException is thrown containing the WebSphere MQ reason code, MQRC_Q_MGR_NOT_AVAILABLE. An error might occur if all the ports in the
specified range are in use, or if a specified IP address, host name, or port number
is not valid; a negative port number, for example.

Example code fragments

This section includes two example code fragments; Figure 1 on page 70 and
Figure 2 on page 73. Each one uses a particular connection and includes notes to
describe the changes needed to use alternative connections.

Example applet code

The following code fragment demonstrates an applet that uses a TCP/IP
connection to:

1. Connect to a queue manager
2. Put a message onto SYSTEM.DEFAULT.LOCAL.QUEUE
3. Get the message back
Example code

```java
// Licensed Materials - Property of IBM
// 5639-C34
// (c) Copyright IBM Corp. 1995,2002
// WebSphere MQ Client for Java sample applet
// This sample runs as an applet using the appletviewer and HTML file,
// using the command :-
//    appletviewer MQSample.html
// Output is to the command line, NOT the applet viewer window.
// Note. If you receive WebSphere MQ error 2 reason 2059 and you are sure your
// WebSphere MQ and TCP/IP setup is correct,
// you should click on the "Applet" selection in the Applet viewer window
// select properties, and change "Network access" to unrestricted.
import com.ibm.mq.*;
// Include the WebSphere MQ classes for Java package
public class MQSample extends java.applet.Applet
{
    private String hostname = "your_hostname";
    // define the name of your
    // host to connect to

    private String channel = "server_channel";
    // define name of channel
    // for client to use
    // Note. assumes WebSphere MQ Server
    // is listening on the default
    // TCP/IP port of 1414

    private String qManager = "your_Q_manager";
    // define name of queue
    // manager object to
    // connect to.

    private MQQueueManager qMgr;
    // define a queue manager object

    // When the class is called, this initialization is done first.
    public void init()
    {
        // Set up WebSphere MQ environment
        MQEnvironment.hostname = hostname;
        // Could have put the
        // hostname & channel
        MQEnvironment.channel = channel;
        // string directly here!

        MQEnvironment.properties.put(MQC.TRANSPORT_PROPERTY,
                                      // Set TCP/IP or server
                                      MQC.TRANSPORT_MQSERIES); //Connection
    }
    // end of init

Figure 1. WebSphere MQ classes for Java example applet (Part 1 of 3)
```
public void start()
{
    try {
        // Create a connection to the queue manager
        qMgr = new MQQueueManager(qManager);

        // Set up the options on the queue we wish to open...
        // Note. All WebSphere MQ Options are prefixed with MQC in Java.
        int openOptions = MQC.MQOO_INPUT_AS_Q_DEF |
                          MQC.MQOO_OUTPUT;
        // Now specify the queue that we wish to open, and the open options...
        MQQueue system_default_local_queue =
            qMgr.accessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                             openOptions);

        // Define a simple WebSphere MQ message, and write some text in UTF format..
        MQMessage hello_world = new MQMessage();
        hello_world.writeUTF("Hello World!");

        // specify the message options...
        MQPutMessageOptions pmo = new MQPutMessageOptions(); // accept the defaults,
                                                               // same as
                                                               // MQPMO_DEFAULT
                                                               // constant

        // put the message on the queue
        system_default_local_queue.put(hello_world, pmo);

        // get the message back again...
        // First define a WebSphere MQ message buffer to receive the message into..
        MQMessage retrievedMessage = new MQMessage();
        retrievedMessage.messageId = hello_world.messageId;

        // Set the get message options..
        MQGetMessageOptions gmo = new MQGetMessageOptions(); // accept the defaults
                                                               // same as
                                                               // MQGMO_DEFAULT

        // get the message off the queue..
        system_default_local_queue.get(retrievedMessage, gmo);

        // And prove we have the message by displaying the UTF message text
        String msgText = retrievedMessage.readUTF();
        System.out.println("The message is: "+ msgText);

        // Close the queue
        system_default_local_queue.close();

        // Disconnect from the queue manager
        qMgr.disconnect();
    }

    // If an error has occurred in the above, try to identify what went wrong.
    // Was it a WebSphere MQ error?
}

Figure 1. WebSphere MQ classes for Java example applet (Part 2 of 3)
Example code

catch (MQException ex)
{
    System.out.println("A WebSphere MQ error occurred: Completion code " +
    ex.completionCode +
    " Reason code " + ex.reasonCode);
}
// Was it a Java buffer space error?
catch (java.io.IOException ex)
{
    System.out.println("An error occurred whilst writing to the
    message buffer: " + ex);
}

} // end of start

} // end of sample

Figure 1. WebSphere MQ classes for Java example applet (Part 3 of 3)

Example application code

The following code fragment demonstrates an application that uses bindings mode to:
1. Connect to a queue manager
2. Put a message onto SYSTEM.DEFAULT.LOCAL.QUEUE
3. Get the message back again
Example code

```java
public class MQSample {
    private String qManager = "your_Q_manager"; // define name of queue
    // manager to connect to.
    private MQQueueManager qMgr; // define a queue manager
    // object
    public static void main(String args[]) {
        new MQSample();
    }
    public MQSample() {
        try {
            // Create a connection to the queue manager
            qMgr = new MQQueueManager(qManager);
            // Set up the options on the queue we wish to open...
            // Note. All WebSphere MQ Options are prefixed with MQC in Java.
            int openOptions = MQC.MQOO_INPUT_AS_Q_DEF |
                              MQC.MQOO_OUTPUT ;
            // Now specify the queue that we wish to open,
            // and the open options...
            MQQueue system_default_local_queue =
                qMgr.accessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                                openOptions);
            // Define a simple WebSphere MQ message, and write some text in UTF format..
            MQMessage hello_world = new MQMessage();
            hello_world.writeUTF("Hello World!");
            // specify the message options...
            MQPutMessageOptions pmo = new MQPutMessageOptions(); // accept the // defaults,
            // same as MQPMO_DEFAULT
```

Figure 2. WebSphere MQ classes for Java example application (Part 1 of 2)
Queue manager operations

// put the message on the queue
system_default_local_queue.put(hello_world, pmo);

// get the message back again...
// First define a WebSphere MQ message buffer to receive the message into...
MQMessage retrievedMessage = new MQMessage();
retrievedMessage.messageId = hello_world.messageId;

// Set the get message options...
MQGetMessageOptions gmo = new MQGetMessageOptions();// accept the defaults
// same as MQGMO_DEFAULT

// get the message off the queue...
system_default_local_queue.get(retrievedMessage, gmo);

// And prove we have the message by displaying the UTF message text
String msgText = retrievedMessage.readUTF();
System.out.println("The message is: " + msgText);

// Close the queue...
system_default_local_queue.close();

// Disconnect from the queue manager
qMgr.disconnect();

// If an error has occurred in the above, try to identify what went wrong
// Was it a WebSphere MQ error?
catch (MQException ex)
{
    System.out.println("A WebSphere MQ error occurred : Completion code "+
ex.completionCode + " Reason code " + ex.reasonCode);
}

// Was it a Java buffer space error?
catch (java.io.IOException ex)
{
    System.out.println("An error occurred whilst writing to the message buffer: " + ex);
}

// end of sample

Figure 2. WebSphere MQ classes for Java example application (Part 2 of 2)

Operations on queue managers

This section describes how to connect to, and disconnect from, a queue manager using WebSphere MQ classes for Java.

Setting up the WebSphere MQ environment

Note: This step is not necessary when using WebSphere MQ classes for Java in bindings mode. In that case, go directly to "Connecting to a queue manager" on page 75. Before you use the client connection to connect to a queue manager, you must set up the MQEnvironment.

The C based WebSphere MQ clients rely on environment variables to control the behavior of the MQCONN call. Because Java applets have no access to environment variables, the Java programming interface includes a class MQEnvironment. This class allows you to specify the following details that are to be used during the connection attempt:
Queue manager operations

- Channel name
- Host name
- Port number
- User ID
- Password

To specify the channel name and host name, use the following code:

```java
MQEnvironment.hostname = "host.domain.com";
MQEnvironment.channel = "java.client.channel";
```

This is equivalent to an MQSERVER environment variable setting of:

"java.client.channel/TCP/host.domain.com".

By default, the Java clients attempt to connect to a WebSphere MQ listener at port 1414. To specify a different port, use the code:

```java
MQEnvironment.port = nnnn;
```

The user ID and password default to blanks. To specify a non-blank user ID or password, use the code:

```java
MQEnvironment.userID = "uid"; // equivalent to env var MQ_USER_ID
MQEnvironment.password = "pwd"; // equivalent to env var MQ_PASSWORD
```

### Connecting to a queue manager

You are now ready to connect to a queue manager by creating a new instance of the MQQueueManager class:

```java
MQQueueManager queueManager = new MQQueueManager("qMgrName");
```

To disconnect from a queue manager, call the disconnect() method on the queue manager:

```java
queueManager.disconnect();
```

If you call the disconnect method, all open queues and processes that you have accessed through that queue manager are closed. However, it is good programming practice to close these resources explicitly when you finish using them. To do this, use the close() method.

The commit() and backout() methods on a queue manager replace the MQCMIT and MQBACK calls that are used with the procedural interface.

### Accessing queues and processes

To access queues and processes, use the MQQueueManager class. The MQOD (object descriptor structure) is collapsed into the parameters of these methods. For example, to open a queue on a queue manager called queueManager, use the following code:

```java
MQQueue queue = queueManager.accessQueue("qName",
MQC.MQOO_OUTPUT,
"qMgrName",
"dynamicQName",
"altUserId");
```

The `options` parameter is the same as the Options parameter in the MQOPEN call.

The accessQueue method returns a new object of class MQQueue.
Queue and process access

When you have finished using the queue, use the close() method to close it, as in the following example:

```
queue.close();
```

With WebSphere MQ classes for Java, you can also create a queue by using the MQQueue constructor. The parameters are exactly the same as for the accessQueue method, with the addition of a queue manager parameter. For example:

```
MQQueue queue = new MQQueue(queueManager,
   "qName",
   MQC.MQOO_OUTPUT,
   "qMgrName",
   "dynamicQName",
   "altUserId");
```

Constructing a queue object in this way enables you to write your own subclasses of MQQueue.

To access a process, use the accessProcess method in place of accessQueue. This method does not have a dynamic queue name parameter, because this does not apply to processes.

The accessProcess method returns a new object of class MQProcess.

When you have finished using the process object, use the close() method to close it, as in the following example:

```
process.close();
```

With WebSphere MQ classes for Java, you can also create a process by using the MQProcess constructor. The parameters are exactly the same as for the accessProcess method, with the addition of a queue manager parameter. Constructing a process object in this way enables you to write your own subclasses of MQProcess.

Handling messages

Put messages onto queues using the put() method of the MQQueue class. You get messages from queues using the get() method of the MQQueue class. Unlike the procedural interface, where MQPUT and MQGET put and get arrays of bytes, the Java programming language puts and gets instances of the MQMessage class. The MQMessage class encapsulates the data buffer that contains the actual message data, together with all the MQMD (message descriptor) parameters that describe that message.

To build a new message, create a new instance of the MQMessage class, and use the writeXXX methods to put data into the message buffer.

When the new message instance is created, all the MQMD parameters are automatically set to their default values, as defined in the WebSphere MQ Application Programming Reference. The put() method of MQQueue also takes an instance of the MQPutMessageOptions class as a parameter. This class represents the MQPMO structure. The following example creates a message and puts it onto a queue:

```
// Build a new message containing my age followed by my name
MQMessage myMessage = new MQMessage();
myMessage.writeInt(25);
String name = "Charlie Jordan";
```
myMessage.writeInt(name.length());
myMessage.writeBytes(name);

// Use the default put message options...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// put the message!
queue.put(myMessage, pmo);

The get() method of MQQueue returns a new instance of MQMessage, which represents the message just taken from the queue. It also takes an instance of the MQGetMessageOptions class as a parameter. This class represents the MQGMO structure.

You do not need to specify a maximum message size, because the get() method automatically adjusts the size of its internal buffer to fit the incoming message. Use the readXXX methods of the MQMessage class to access the data in the returned message.

The following example shows how to get a message from a queue:

```
// Get a message from the queue
MQMessage theMessage = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.get(theMessage, gmo); // has default values

// Extract the message data
int age = theMessage.readInt();
int strLen = theMessage.readInt();
byte[] strData = new byte[strLen];
theMessage.readFully(strData, 0, strLen);
String name = new String(strData, 0);
```

You can alter the number format that the read and write methods use by setting the encoding member variable.

You can alter the character set to use for reading and writing strings by setting the charset member variable.

See "MQMessage" on page 126 for more details.

**Note:** The writeUTF() method of MQMessage automatically encodes the length of the string as well as the Unicode bytes it contains. When your message will be read by another Java program (using readUTF()), this is the simplest way to send string information.

### Handling errors

Methods in the Java interface do not return a completion code and reason code. Instead, they throw an exception whenever the completion code and reason code resulting from a WebSphere MQ call are not both zero. This simplifies the program logic so that you do not have to check the return codes after each call to WebSphere MQ. You can decide at which points in your program you want to deal with the possibility of failure. At these points, you can surround your code with try and catch blocks, as in the following example:

```
try {
   myQueue.put(messageA, putMessageOptionsA);
   myQueue.put(messageB, putMessageOptionsB);
} catch (MQException ex) {
   // handle exception
}
```
Error handling

// This block of code is only executed if one of
// the two put methods gave rise to a non-zero
// completion code or reason code.
System.out.println("An error occurred during the put operation:" +
   "CC = " + ex.completionCode +
   "RC = " + ex.reasonCode);
System.out.println("Cause exception:" + ex.getCause() );
}

The WebSphere MQ call reason codes reported back in Java exceptions are documented in a chapter called “Return Codes” in the WebSphere MQ Application Programming Reference.

Sometimes the reason code does not convey all details associated with the error. This can occur if WebSphere MQ uses services provided by another product (for example, a JSSE implementation) that throws a java.lang.Exception to WebSphere MQ Java. In this case, the method MQException.getCause() retrieves the underlying java.lang.Exception that caused the error.

Getting and setting attribute values

For many of the common attributes, the classes MQManagedObject, MQQueue, MQProcess, and MQQueueManager contain getXXX() and setXXX() methods. These methods allow you to get and set their attribute values. Note that for MQQueue, the methods work only if you specify the appropriate inquire and set flags when you open the queue.

For less common attributes, the MQQueueManager, MQQueue, and MQProcess classes all inherit from a class called MQManagedObject. This class defines the inquire() and set() interfaces.

When you create a new queue manager object by using the new operator, it is automatically opened for inquire. When you use the accessProcess() method to access a process object, that object is automatically opened for inquire. When you use the accessQueue() method to access a queue object, that object is not automatically opened for either inquire or set operations. This is because adding these options automatically can cause problems with some types of remote queues. To use the inquire, set, getXXX, and setXXX methods on a queue, you must specify the appropriate inquire and set flags in the openOptions parameter of the accessQueue() method.

The inquire and set methods take three parameters:
* selectors array
* intAttrs array
* charAttrs array

You do not need the SelectorCount, IntAttrCount, and CharAttrLength parameters that are found in MQINQ, because the length of an array in Java is always known. The following example shows how to make an inquiry on a queue:

// inquire on a queue
final static int MQIA_DEF_PRIORITY = 6;
final static int MQCA_Q_DESC = 2013;
final static int MQ_Q_DESC_LENGTH = 64;

int[] selectors = new int[2];
int[] intAttrs = new int[2];
byte[] charAttrs = new byte[MQ_Q_DESC_LENGTH]

selectors[0] = MQIA_DEF_PRIORITY;
selectors[1] = MQCA_Q_DESC;
queue.inquire(selectors,intAttrs,charAttrs);
System.out.println("Default Priority = " + intAttrs[0]);
System.out.println("Description : " + new String(charAttrs,0));

Multithreaded programs

Multithreaded programs are hard to avoid in Java. Consider a simple program that
connects to a queue manager and opens a queue at startup. The program displays
a single button on the screen. When a user presses that button, the program fetches
a message from the queue.

The Java runtime environment is inherently multithreaded. Therefore, your
application initialization occurs in one thread, and the code that executes in
response to the button press executes in a separate thread (the user interface
thread).

With the C based WebSphere MQ client, this would cause a problem, because
handles cannot be shared across multiple threads. WebSphere MQ classes for Java
relaxes this constraint, allowing a queue manager object (and its associated queue
and process objects) to be shared across multiple threads.

The implementation of WebSphere MQ classes for Java ensures that, for a given
connection (MQQueueManager object instance), all access to the target WebSphere
MQ queue manager is synchronized. A thread that wants to issue a call to a queue
manager is blocked until all other calls in progress for that connection are
complete. If you require simultaneous access to the same queue manager from
multiple threads within your program, create a new MQQueueManager object for
each thread that requires concurrent access. (This is equivalent to issuing a
separate MQCONN call for each thread.)

Writing user exits

WebSphere MQ classes for Java allows you to provide your own send, receive, and
security exits.

To implement an exit, you define a new Java class that implements the appropriate
interface. Three exit interfaces are defined in the WebSphere MQ package:
- MQSendExit
- MQReceiveExit
- MQSecurityExit

Note: User exits are supported for client connections only; they are not supported
for bindings connections.

Any SSL encryption defined for a connection is performed after the send exit has
been invoked. Similarly, decryption is performed before the receive or security exits
are invoked.

The following sample defines a class that implements all three:
class MyMQExits implements MQSendExit, MQReceiveExit, MQSecurityExit {
    // This method comes from the send exit
    public byte[] sendExit(MQChannelExit channelExitParms,
                            MQChannelDefinition channelDefParms,
                            string[]... channelParms) {
        // Implementation code
    }
}
Writing user exits

```java
byte agentBuffer[])
{
  // fill in the body of the send exit here
}

// This method comes from the receive exit
public byte[] receiveExit(MQChannelExit channelExitParms,
                          MQChannelDefinition channelDefParms,
                          byte agentBuffer[])
{
  // fill in the body of the receive exit here
}

// This method comes from the security exit
public byte[] securityExit(MQChannelExit channelExitParms,
                           MQChannelDefinition channelDefParms,
                           byte agentBuffer[])
{
  // fill in the body of the security exit here
}
}
```

Each exit is passed an MQChannelExit and an MQChannelDefinition object instance. These objects represent the MQCXP and MQCD structures defined in the procedural interface.

For a Send exit, the `agentBuffer` parameter contains the data that is about to be sent. For a Receive exit or a Security exit, the `agentBuffer` parameter contains the data that has just been received. You do not need a length parameter, because the expression `agentBuffer.length` indicates the length of the array.

For the Send and Security exits, your exit code should return the byte array that you want to send to the server. For a Receive exit, your exit code must return the modified data that you want WebSphere MQ classes for Java to interpret.

The simplest possible exit body is:
```java
{
  return agentBuffer;
}
```

If your program is to run as a downloaded Java applet, the security restrictions that apply mean that you cannot read or write any local files. If your exit needs a configuration file, you can place the file on the Web and use the java.net.URL class to download it and examine its contents.

Connection pooling

WebSphere MQ classes for Java provides additional support for applications that deal with multiple connections to WebSphere MQ queue managers. When a connection is no longer required, instead of destroying it, it can be pooled and later reused. This can provide a substantial performance enhancement for applications and middleware that connect serially to arbitrary queue managers.

WebSphere MQ provides a default connection pool. Applications can activate or deactivate this connection pool by registering and deregistering tokens through the MQEnvironment class. If the pool is active when WebSphere MQ base Java constructs an MQQueueManager object, it searches this default pool and reuses
any suitable connection. When an MQQueueManager.disconnect() call occurs, the underlying connection is returned to the pool.

Alternatively, applications can construct an MQSimpleConnectionManager connection pool for a particular use. Then, the application can either specify that pool during construction of an MQQueueManager object, or pass that pool to MQEnvironment for use as the default connection pool.

To prevent connections from using too much resource, you can limit the total number of connections that an MQSimpleConnectionManager object can handle, and you can limit the size of the connection pool. Setting limits is useful if there are conflicting demands for connections within a JVM.

By default, the getMaxConnections() method returns the value zero, which means that there is no limit to the number of connections that the MQSimpleConnectionManager object can handle. You can set a limit by using the setMaxConnections() method. If you set a limit and the limit is reached, a request for a further connection might cause an MQException to be thrown, with a reason code of MQRC_MAX_CONNS_LIMIT_REACHED.

Also, WebSphere MQ base Java provides a partial implementation of the Java 2 Platform Enterprise Edition (J2EE) Connector Architecture. Applications running under a Java 2 v1.3 JVM with JAAS 1.0 (Java Authentication and Authorization Service) can provide their own connection pool by implementing the javax.resource.spi.ConnectionManager interface. Again, this interface can be specified on the MQQueueManager constructor, or specified as the default connection pool.

**Controlling the default connection pool**

Consider the following example application, MQApp1:

```java
import com.ibm.mq.*;
public class MQApp1 {
    public static void main(String[] args) throws MQException {
        for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            : (do something with qmgr)
            qmgr.disconnect();
        }
    }
}
```

MQApp1 takes a list of local queue managers from the command line, connects to each in turn, and performs some operation. However, when the command line lists the same queue manager many times, it is more efficient to connect only once, and to reuse that connection many times.

WebSphere MQ base Java provides a default connection pool that you can use to do this. To enable the pool, use one of the MQEnvironment.addConnectionPoolToken() methods. To disable the pool, use MQEnvironment.removeConnectionPoolToken().

The following example application, MQApp2, is functionally identical to MQApp1, but connects only once to each queue manager.
Connection pooling

```java
import com.ibm.mq.*;
public class MQApp2 {
    public static void main(String[] args) throws MQException {
        MQPoolToken token = MQEnvironment.addConnectionPoolToken();
        for (int i = 0; i < args.length; i++) {
            MQQueueManager qmgr = new MQQueueManager(args[i]);
                : (do something with qmgr)
                : qmgr.disconnect();
        }
        MQEnvironment.removeConnectionPoolToken(token);
    }
}
```

The first bold line activates the default connection pool by registering an MQPoolToken object with MQEnvironment.

The MQQueueManager constructor now searches this pool for an appropriate connection and only creates a connection to the queue manager if it cannot find an existing one. The qmgr.disconnect() call returns the connection to the pool for later reuse. These API calls are the same as the sample application MQApp1.

The second highlighted line deactivates the default connection pool, which destroys any queue manager connections stored in the pool. This is important because otherwise the application would terminate with a number of live queue manager connections in the pool. This situation could cause errors that would appear in the queue manager logs.

The default connection pool stores a maximum of ten unused connections, and keeps unused connections active for a maximum of five minutes. The application can alter this (for details, see "Supplying a different connection pool" on page 84).

Instead of using MQEnvironment to supply an MQPoolToken, the application can construct its own:

```java
MQPoolToken token = new MQPoolToken();
MQEnvironment.addConnectionPoolToken(token);
```

Some applications or middleware vendors provide subclasses of MQPoolToken in order to pass information to a custom connection pool. They can be constructed and passed to addConnectionPoolToken() in this way so that extra information can be passed to the connection pool.
The default connection pool and multiple components

MQEnvironment holds a static set of registered MQPoolToken objects. To add or remove MQPoolTokens from this set, use the following methods:

- MQEnvironment.addConnectionPoolToken()
- MQEnvironment.removeConnectionPoolToken()

An application might consist of many components that exist independently and perform work using a queue manager. In such an application, each component should add an MQPoolToken to the MQEnvironment set for its lifetime.

For example, the example application MQApp3 creates ten threads and starts each one. Each thread registers its own MQPoolToken, waits for a length of time, then connects to the queue manager. After the thread disconnects, it removes its own MQPoolToken.

The default connection pool remains active while there is at least one token in the set of MQPoolTokens, so it will remain active for the duration of this application. The application does not need to keep a master object in overall control of the threads.

```java
import com.ibm.mq.*;
public class MQApp3 {
    public static void main(String[] args) {
        for (int i=0; i<10; i++) {
            MQApp3_Thread thread=new MQApp3_Thread(i*60000);
            thread.start();
        }
    }
}

class MQApp3_Thread extends Thread {
    long time;
    public MQApp3_Thread(long time) {
        this.time=time;
    }
    public synchronized void run() {
        MQPoolToken token=MQEnvironment.addConnectionPoolToken();
        try {
            wait(time);
            MQQueueManager qmgr=new MQQueueManager("my.qmgr.1");
            : (do something with qmgr)
            qmgr.disconnect();
        } catch (MQException mqe) {System.err.println("Error occurred!");}
        catch (InterruptedException ie) {} 
        MQEnvironment.removeConnectionPoolToken(token);
    }
}
```
Supplying a different connection pool

This section describes how to use the class `com.ibm.mq.MQSimpleConnectionManager` to supply a different connection pool. This class provides basic facilities for connection pooling, and applications can use this class to customize the behavior of the pool.

Once it is instantiated, an MQSimpleConnectionManager can be specified on the MQQueueManager constructor. The MQSimpleConnectionManager then manages the connection that underlies the constructed MQQueueManager. If the MQSimpleConnectionManager contains a suitable pooled connection, that connection is reused and returned to the MQSimpleConnectionManager after an MQQueueManager.disconnect() call.

The following code fragment demonstrates this behavior:

```java
MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
myConnMan.setActive(MQSimpleConnectionManager.MODE_ACTIVE);
MQQueueManager qmgr=new MQQueueManager("my.qmgr.1", myConnMan);
: (do something with qmgr)
: qmgr.disconnect();

MQQueueManager qmgr2=new MQQueueManager("my.qmgr.1", myConnMan);
: (do something with qmgr2)
: qmgr2.disconnect();
myConnMan.setActive(MQSimpleConnectionManager.MODE_INACTIVE);
```

The connection that is forged during the first MQQueueManager constructor is stored in myConnMan after the qmgr.disconnect() call. The connection is then reused during the second call to the MQQueueManager constructor.

The second line enables the MQSimpleConnectionManager. The last line disables MQSimpleConnectionManager, destroying any connections held in the pool. An MQSimpleConnectionManager is, by default, in MODE_AUTO, which is described later in this section.

An MQSimpleConnectionManager allocates connections on a most-recently-used basis, and destroys connections on a least-recently-used basis. By default, a connection is destroyed if it has not been used for five minutes, or if there are more than ten unused connections in the pool. You can alter these values using:

- `MQSimpleConnectionManager.setTimeout()`
- `MQSimpleConnectionManager.setHighThreshold()`

You can also set up an MQSimpleConnectionManager for use as the default connection pool, to be used when no Connection Manager is supplied on the MQQueueManager constructor.
The following application demonstrates this:

```java
import com.ibm.mq.*;
public class MQApp4 {
    public static void main(String[] args) {
        MQSimpleConnectionManager myConnMan = new MQSimpleConnectionManager();
        myConnMan.setActive(MQSimpleConnectionManager.MODE_AUTO);
        myConnMan.setTimeout(3600000);
        myConnMan.setMaxConnections(75);
        myConnMan.setMaxUnusedConnections(50);
        MQEnvironment.setDefaultConnectionManager(myConnMan);
        MQApp3.main(args);
    }
}
```

The bold lines create and configure an MQSimpleConnectionManager object. The configuration does the following:

- Ends connections that are not used for an hour
- Limits the number of connections managed by myConnMan to 75
- Limits the number of unused connections in the pool to 50
- Sets MODE_AUTO, which is the default. This means that the pool is active only if it is the default connection manager, and there is at least one token in the set of MQPoolTokens held by MQEnvironment.

The new MQSimpleConnectionManager is then set as the default connection manager.

In the last line, the application calls MQApp3.main(). This runs a number of threads, where each thread uses WebSphere MQ independently. These threads use myConnMan when they forge connections.

**Supplying your own ConnectionManager**

Under Java 2 v1.3, with JAAS 1.0 installed, applications and middleware providers can provide alternative implementations of connection pools. WebSphere MQ base Java provides a partial implementation of the J2EE Connector Architecture. Implementations of `javax.resource.spi.ConnectionManager` can either be used as the default Connection Manager or be specified on the MQQueueManager constructor.

WebSphere MQ base Java complies with the Connection Management contract of the J2EE Connector Architecture. Read this section in conjunction with the Connection Management contract of the J2EE Connector Architecture (refer to Sun’s Web site at http://java.sun.com).

The ConnectionManager interface defines only one method:

```java
package javax.resource.spi;
public interface ConnectionManager {
    Object allocateConnection(ManagedConnectionFactory mcf,
                               ConnectionRequestInfo cxRequestInfo);
}
```

The MQQueueManager constructor calls allocateConnection on the appropriate ConnectionManager. It passes appropriate implementations of ManagedConnectionFactory and ConnectionRequestInfo as parameters to describe the connection required.
Connection pooling

The ConnectionManager searches its pool for a javax.resource.spi.ManagedConnection object that has been created with identical ManagedConnectionFactory and ConnectionRequestInfo objects. If the ConnectionManager finds any suitable ManagedConnection objects, it creates a java.util.Set that contains the candidate ManagedConnections. Then, the ConnectionManager calls the following:

ManagedConnection mc=mcf.matchManagedConnections(connectionSet, subject, cxRequestInfo);

The WebSphere MQ implementation of ManagedConnectionFactory ignores the subject parameter. This method selects and returns a suitable ManagedConnection from the set, or returns null if it does not find a suitable ManagedConnection. If there is not a suitable ManagedConnection in the pool, the ConnectionManager can create one by using:

ManagedConnection mc=mcf.createManagedConnection(subject, cxRequestInfo);

Again, the subject parameter is ignored. This method connects to a WebSphere MQ queue manager and returns an implementation of javax.resource.spi.ManagedConnection that represents the newly-forged connection. Once the ConnectionManager has obtained a ManagedConnection (either from the pool or freshly created), it creates a connection handle using:

Object handle=mc.getConnection(subject, cxRequestInfo);

This connection handle can be returned from allocateConnection().

A ConnectionManager must register an interest in the ManagedConnection through:

mc.addConnectionEventListener()

The ConnectionEventListener is notified if a severe error occurs on the connection, or when MQQueueManager.disconnect() is called. When MQQueueManager.disconnect() is called, the ConnectionEventListener can do either of the following:

- Reset the ManagedConnection using the mc.cleanup() call, then return the ManagedConnection to the pool
- Destroy the ManagedConnection using the mc.destroy() call

If the ConnectionManager is the default ConnectionManager, it can also register an interest in the state of the MQEnvironment-managed set of MQPoolTokens. To do so, first construct an MQPoolServices object, then register an MQPoolServicesEventListener object with the MQPoolServices object:

MQPoolServices maps=new MQPoolServices();
maps.addMQPoolServicesEventListener(listener);

The listener is notified when an MQPoolToken is added or removed from the set, or when the default ConnectionManager changes. The MQPoolServices object also provides a way to query the current size of the set of MQPoolTokens.
JTA/JDBC coordination using WebSphere MQ base Java

WebSphere MQ base Java supports the MQQueueManager.begin() method, which allows WebSphere MQ to act as a coordinator for a database which provides a JDBC 2 compliant driver. Currently this support is available on Solaris, AIX, and Windows systems with Oracle or DB2 databases.

Installation

In order to use the XA-JTA support, you must use the special JTA switch library. The method for using this library varies depending on whether you are using Windows systems or one of the other platforms.

Installation on Windows systems

On Windows systems, the new XA library is supplied as a complete DLL. The name of this DLL is jdbcxxxx.dll where xxx indicates the database for which the switch library has been compiled. This library is in the java/1ib/jdbc directory of your WebSphere MQ base Java installation.

Installation on other platforms

The switch file is supplied as an object file that you must link yourself using the supplied makefile. This is necessary because certain libraries required by the switch library might be in different locations on different systems. Because the switch library is loaded by the queue manager, which runs in a setuid environment, you cannot use the dynamic library path variable to locate these libraries. You therefore need to put the full path names to these libraries in the switch library itself.

The object files are called jdbcxxxx.o where xxx indicates which database the object file is for. When linked, a switch file called jdbcxxx is produced; add this to the qm.ini file in the same manner as the standard switch libraries.

To create the switch library, go into the java/1ib/jdbc subdirectory of your WebSphere MQ base Java installation and run make with your target database as a parameter. Currently supported targets for XA-JTA are oracle and db2. For example:

```
make db2
```

The makefiles are set up to link against the databases and JDKs in their standard installed location. The exception to this is Oracle, which can be installed anywhere on the system. The makefile uses Oracle’s ORACLE_HOME environment variable to link the library correctly. If your JDK is in a non-standard location, you can override the default directory with the JAVA_HOME definition:

```
make JAVA_HOME=/usr/my_jdk13 oracle
```

The above command produces a switch file named jdbcora, which is used in the same way as a standard switch library, including using the same XAOpenString. If you have previously configured an XAResourceManager in your qm.ini, replace the SwitchFile line with a reference to the new JTA-specific switch file. If you have not previously used an XA switch file, refer to the [WebSphere MQ System Administration Guide](#) for of configuring the XAResourceManager stanza for different databases, remembering to replace the standard switch file with the Java-specific one.

Once you have updated the qm.ini, restart the queue manager. Ensure that all appropriate database environment variables have been set before calling strmqm.
Usage

The basic sequence of API calls for a user application is:
```java
qMgr = new MQQueueManager("QM1")
Connection con = qMgr.getJDBCConnection( xads );
qMgr.begin()

< Perform MQ and DB operations to be grouped in a unit of work >

qMgr.commit() or qMgr.backout();
con.close();
qMgr.disconnect()
```
xads in the getJDBCConnection call is a database-specific implementation of the XADataSource interface, which defines the details of the database to connect to. See the documentation for your database to determine how to create an appropriate XADataSource object to pass into getJDBCConnection.

You also need to update your CLASSPATH with the appropriate database-specific jar files for performing JDBC work.

If you need to connect to multiple databases, you might have to call getJDBCConnection several times to perform the transaction across several different connections.

There are two forms of the getJDBCConnection, reflecting the two forms of XADataSource.getXAConnection:
```java
public java.sql.Connection getJDBCConnection(javax.sql.XADataSource xads)
  throws MQException, SQLException, Exception

public java.sql.Connection getJDBCConnection(XADataSource dataSource, String userid, String password)
  throws MQException, SQLException, Exception
```

These methods declare Exception in their throws clauses to avoid problems with the JVM verifier for customers who are not using the JTA functionality. The actual exception thrown is javax.transaction.xa.XAException, which requires the jta.jar file to be added to the classpath for programs that did not previously require it.

Known problems and limitations

Because this support makes calls to JDBC drivers, the implementation of those JDBC drivers can have significant impact on the system behavior. In particular, tested JDBC drivers behave differently when the database is shut down while an application is running. **Always** avoid abruptly shutting down a database while there are applications holding open connections to it.

Oracle 8.1.7

Calling the JDBC Connection.close() method after MQQueueManager.disconnect() generates an SQLException. Either call Connection.close() before MQQueueManager.disconnect(), or omit the call to Connection.close().

DB2® Sometimes DB2 returns a SQL0805N error. This problem can be resolved with the following CLP command:
```
DB2 bind @db2cli.lst blocking all grant public
```

Refer to the DB2 documentation for more information.
Solaris and JDK 1.3

When running on Solaris with JDK 1.3, attempting to start the queue manager from a user ID other than root or mqm is likely to fail to load the jdbcora switch file with the following message being put to the error log:

AMQ6175: The system could not dynamically load the library /opt/mqm/java/lib/jdbcora. The error message was ld.so.1: amqzma0: fatal: libverify.so: open failed: No such file or directory. The Queue Manager will continue without this module.

The problem arises because of a dependency between two libraries in JDK1.3 that cannot be resolved by the dynamic linker when the invoking program has the setuid bit set (as strmqm does).

Under these circumstances, start the queue manager from the mqm user ID; if this is not practical, make a symbolic link for libverify into /usr/lib. For example:

```
ln -s /usr/j2se/jre/lib/sparc/libverify.so /usr/lib/libverify.so
```

Solaris and multiple XAResourceManager stanzas

When attempting to use multiple XAResourceManager stanzas on any given queue manager on Solaris, the commit call might fail. Treat this as an unsupported combination; it does not affect queue managers with a single XAResourceManager stanza.

Windows systems

The JDBC libraries supplied with WebSphere MQ Java (jdbcdb2.dll and jdbcora.dll) have a dependency on jvm.dll, which is supplied with the JVM. However, depending on the JVM used, this DLL might be in a subdirectory that is not on the path; for example, jre/bin/classic/jvm.dll.

If jvm.dll cannot be found when the queue manager starts, the queue manager produces a message like the following (this example is for DB2):

AMQ6174: The library C:\Program Files\IBM\MQSeries\Java\lib\jdbc\jdbcdb2.dll was not found. The queue manager will continue without this module.

In fact, the file not found is jvm.dll. The solution is to either copy jvm.dll to somewhere already on the path or update the path to include the location of jvm.dll.

Secure Sockets Layer (SSL) support

WebSphere MQ base Java client applications and WebSphere MQ JMS connections using TRANSPORT(CLIENT) support Secure Sockets Layer (SSL) encryption. SSL provides communication encryption, authentication, and message integrity. It is typically used to secure communications between any two peers on the Internet or within an intranet.

WebSphere MQ classes for Java uses Java Secure Socket Extension (JSSE) to handle SSL encryption, and so requires a JSSE provider. J2SE v1.4 JVMs have a JSSE provider built in. Details of how to manage and store certificates can vary from provider to provider. For information about this, refer to your JSSE provider’s documentation.

This section assumes that your JSSE provider is correctly installed and configured, and that suitable certificates have been installed and made available to your JSSE provider.
SSL support

Enabling SSL

SSL is supported only for client connections. To enable SSL, you must specify the CipherSuite to use when communicating with the queue manager, and this must match the CipherSpec set on the target channel. Additionally, the named CipherSuite must be supported by your JSSE provider. However, CipherSuites are distinct from CipherSpecs and so have different names. Appendix H, “SSL CipherSuites supported by WebSphere MQ” on page 487 contains a table mapping the CipherSpecs supported by WebSphere MQ to their equivalent CipherSuites as known to JSSE.

To enable SSL, specify the CipherSuite using the sslCipherSuite static member variable of MQEnvironment. The following example attaches to a SVRCONN channel named SECURE.SVRCONN.CHANNEL, which has been set up to require SSL with a CipherSpec of RC4_MD5_EXPORT:

```java
MQEnvironment.hostname = "your_hostname";
MQEnvironment.channel = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.sslCipherSuite = "SSL_RSA_EXPORT_WITH_RC4_40_MD5";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

Note that, although the channel has a CipherSpec of RC4_MD5_EXPORT, the Java application must specify a CipherSuite of SSL_RSA_EXPORT_WITH_RC4_40_MD5. For more information about CipherSpecs and CipherSuites, see the [WebSphere MQ Security] book. See Appendix H, “SSL CipherSuites supported by WebSphere MQ” on page 487 for a list of mappings between CipherSpecs and CipherSuites.

The sslCipherSuite property can also be set using the MQC.SSL_CIPHER_SUITEPROPERTY in the Hash table of connection properties.

To successfully connect using SSL, the JSSE TrustStore must be set up with Certificate Authority root certificates from which the certificate presented by the queue manager can be authenticated. Similarly, if SSLClientAuth on the SVRCONN channel has been set to MQSSL_CLIENT_AUTH_REQUIRED, the JSSE KeyStore must contain an identifying certificate that is trusted by the queue manager.

Using the distinguished name of the queue manager

The queue manager identifies itself using an SSL certificate, which contains a Distinguished Name (DN). A WebSphere MQ Java client application can use this DN to ensure that it is communicating with the correct queue manager. A DN pattern is specified using the sslPeerName variable of MQEnvironment. For example, setting:

```java
MQEnvironment.sslPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

allows the connection to succeed only if the queue manager presents a certificate with a Common Name beginning QMGR, and at least two Organizational Unit names, the first of which must be IBM and the second WEBSPHERE.

The sslPeerName property can also be set using the MQC.SSL_PEER_NAME_PROPERTY in the hash table of connection properties. For more information about distinguished names, refer to [WebSphere MQ Security].

If sslPeerName is set, connections succeed only if it is set to a valid pattern and the queue manager presents a matching certificate.
Using certificate revocation lists

A certificate revocation list (CRL) is a set of certificates that have been revoked, either by the issuing Certificate Authority or by the local organization. CRLs are typically hosted on LDAP servers. With Java 2 v1.4, a CRL server can be specified at connect-time and the certificate presented by the queue manager is checked against the CRL before the connection is allowed. For more information about certificate revocation lists and WebSphere MQ, see [WebSphere MQ Security](#).

Note: To use a CertStore successfully with a CRL hosted on an LDAP server, make sure that your Java Software Development Kit (SDK) is compatible with the CRL. Some SDKs require that the CRL conforms to RFC 2587, which defines a schema for LDAP v2. Most LDAP v3 servers use RFC 2256 instead.

The CRLs to use are specified through the java.security.cert.CertStore class. Refer to documentation on this class for full details of how to obtain instances of CertStore. To create a CertStore based on an LDAP server, first create an LDAPCertStoreParameters instance, initialized with the server and port settings to use. For example:

```java
import java.security.cert.*;
CertStoreParameters csp = new LDAPCertStoreParameters("crl_server", 389);
```

Having created a CertStoreParameters instance, use the static constructor on CertStore to create a CertStore of type LDAP:

```java
CertStore cs = CertStore.getInstance("LDAP", csp);
```

Other CertStore types (for example, Collection) are also supported. Commonly there are several CRL servers set up with identical CRL information to give redundancy. Once you have a CertStore object for each of these CRL servers, place them all in a suitable Collection. The following example shows the CertStore objects placed in an ArrayList:

```java
import java.util.ArrayList;
Collection crls = new ArrayList();
crls.add(cs);
```

This Collection can be set into the MQEnvironment static variable, sslCertStores, before connecting to enable CRL checking:

```java
MQEnvironment.sslCertStores = crls;
```

The certificate presented by the queue manager when a connection is being set up is validated as follows:

1. The first CertStore object in the Collection identified by sslCertStores is used to identify a CRL server.
2. An attempt is made to contact the CRL server.
3. If the attempt is successful, the server is searched for a match for the certificate.
   a. If the certificate is found to be revoked, the search process is over and the connection request fails with reason code MQRC_SSL_CERTIFICATE_REVOKED.
   b. If the certificate is not found, the search process is over and the connection is allowed to proceed.
4. If the attempt to contact the server is unsuccessful, the next CertStore object is used to identify a CRL server and the process repeats from step 2.

If this was the last CertStore in the Collection, or if the Collection contains no CertStore objects, the search process has failed and the connection request fails with reason code MQRC_SSL_CERT_STORE_ERROR.
SSL support

The Collection object determines the order in which CertStores are used.

The Collection of CertStores can also be set using the
MQC_SSL_CERT_STORE_PROPERTY. As a convenience, this property also allows a
single CertStore to be specified without needing to be a member of a Collection.

If sslCertStores is set to null, no CRL checking is performed. This property is
ignored if sslCipherSuite is not set.

Supplying a customized SSLSocketFactory

Different JSSE implementations can provide different features. For example, a
specialized JSSE implementation could allow configuration of a particular model of
encryption hardware. Additionally, some JSSE providers allow customization of
KeyStores and TrustStores by program, or allow the choice of identity certificate
from the KeyStore to be altered. In JSSE, all these customizations are abstracted
into a factory class, javax.net.ssl.SSLSocketFactory.

Refer to your JSSE documentation for details of how to create a customized
SSLSocketFactory implementation. The details vary from provider to provider, but
a typical sequence of steps might be:
1. Create an SSLContext object using a static method on SSLContext
2. Initialize this SSLContext with appropriate KeyManagers and TrustManager
   implementations (created from their own factory classes)
3. Create an SSLSocketFactory from the SSLContext

When you have an SSLSocketFactory object, set the
MQEnvironment.sslSocketFactory to the customized factory object. For example:

```java
javax.net.ssl.SSLSocketFactory sf = sslContext.getSocketFactory();
MQEnvironment.sslSocketFactory = sf;
```

WebSphere MQ classes for Java then use this SSLSocketFactory to connect to the
WebSphere MQ queue manager. This property can also be set using the
MQC_SSL_SOCKET_FACTORY_PROPERTY. If ssSocketFactory is set to null, the
JVM's default SSLSocketFactory is used. This property is ignored if sslCipherSuite
is not set.

Error handling when using SSL

The following reason codes can be issued by WebSphere MQ classes for Java when
connecting to a queue manager using SSL:

**MQRC_SSL_NOT_ALLOWED**
The sslCipherSuite property was set, but bindings connect was used. Only
client connect supports SSL.

**MQRC_JSSE_ERROR**
The JSSE provider reported an error that could not be handled by
WebSphere MQ. This could be caused by a configuration problem with
JSSE, or because the certificate presented by the queue manager could not
be validated. The exception produced by JSSE can be retrieved using the
getCause() method on MQException.

**MQRC_SSL_PEER_NAME_MISMATCH**
The DN pattern specified in the sslPeerName property did not match the
DN presented by the queue manager.
MQRC_SSL_PEER_NAME_ERROR
The DN pattern specified in the sslPeerName property was not valid.

MQRC_UNSUPPORTED_CIPHER_SUITE
The CipherSuite named in sslCipherSuite was not recognized by the JSSE provider. A full list of CipherSuites supported by the JSSE provider can be obtained by a program using the SSLSocketFactory.getSupportedCipherSuites() method. A list of CipherSuites that can be used to communicate with WebSphere MQ can be found in Appendix H, “SSL CipherSuites supported by WebSphere MQ,” on page 487.

MQRC_SSL_CERTIFICATE_REVOKED
The certificate presented by the queue manager was found in a CRL specified with the sslCertStores property. Update the queue manager to use trusted certificates.

MQRC_SSL_CERT_STORE_ERROR
None of the supplied CertStores could be searched for the certificate presented by the queue manager. The MQException.getCause() method returns the error that occurred while searching the first CertStore attempted. If the causal exception is NoSuchElementException, ClassCastException, or NullPointerException, check that the Collection specified on the sslCertStores property contains at least one valid CertStore object.

### Compiling and testing WebSphere MQ base Java programs

Before compiling WebSphere MQ base Java programs, you must ensure that your WebSphere MQ classes for Java installation directory is in your CLASSPATH environment variable, as described in Chapter 2, “Installation,” on page 9.

To compile a class called MyClass.java, use the command:

```bash
javac MyClass.java
```

### Running WebSphere MQ base Java applets

If you write an applet (subclass of java.applet.Applet), you must create an HTML file referencing your class before you can run it. A sample HTML file might look as follows:

```html
<html>
<body>
<applet code="MyClass.class" width=200 height=400>
</applet>
</body>
</html>
```

Run your applet either by loading this HTML file into a Java-enabled Web browser, or by using the appletviewer that comes with the Java Development Kit (JDK).

To use the applet viewer, enter the command:

```bash
appletviewer myclass.html
```
Running WebSphere MQ base Java applications

If you write an application (a class that contains a main() method), using either the client or the bindings mode, run your program using the Java interpreter. Use the command:

```
java MyClass
```

**Note:** The .class extension is omitted from the class name.

Tracing WebSphere MQ base Java programs

WebSphere MQ base Java includes a trace facility, which you can use to produce diagnostic messages if you suspect that there might be a problem with the code. (You normally need to use this facility only at the request of IBM service.)

Tracing is controlled by the enableTracing and disableTracing methods of the MQEnvironment class. For example:

```
MQEnvironment.enableTracing(2); // trace at level 2
...                      // these commands will be traced
MQEnvironment.disableTracing(); // turn tracing off again
```

The trace is written to the Java console (System.err).

If your program is an application, or if you run it from your local disk using the appletviewer command, you can also redirect the trace output to a file of your choice. The following code fragment shows an example of how to redirect the trace output to a file called myapp.trc:

```
import java.io.*;

try {
    FileOutputStream traceFile = new FileOutputStream("myapp.trc");
    MQEnvironment.enableTracing(2, traceFile);
} catch (IOException ex) {
    // couldn't open the file,
    // trace to System.err instead
    MQEnvironment.enableTracing(2);
}
```

There are five different levels of tracing:

1. Provides entry, exit, and exception tracing
2. Provides parameter information in addition to 1
3. Provides transmitted and received WebSphere MQ headers and data blocks in addition to 2
4. Provides transmitted and received user message data in addition to 3
5. Provides tracing of methods in the Java Virtual Machine in addition to 4

To trace methods in the Java Virtual Machine with trace level 5:

- For an application, run it by issuing the command `java_g` (instead of `java`)
- For an applet, run it by issuing the command `appletviewer_g` (instead of `appletviewer`)

**Note:** `java_g` is not supported on OS/400, but similar function is provided by using `OPTION(*VERBOSE)` on the RUNJVA command.
Chapter 8. Environment-dependent behavior

WebSphere MQ classes for Java allow you to create applications that can run against different versions of WebSphere MQ and MQSeries. This chapter describes the behavior of the Java classes dependent on these different versions.

WebSphere MQ classes for Java provides a core of classes, which provide consistent function and behavior in all the environments. Features outside this core depend on the capability of the queue manager to which the application is connected.

Except where noted here, the behavior exhibited is as described in the Application Programming Reference book appropriate to the queue manager.

Core details

WebSphere MQ classes for Java contains the following core set of classes, which can be used in all environments with only the minor variations listed in "Restrictions and variations for core classes" on page 96:

- MQEnvironment
- MQException
- MQGetMessageOptions
  Excluding:
  - MatchOptions
  - GroupStatus
  - SegmentStatus
  - Segmentation
- MQManagedObject
  Excluding:
  - inquire()
  - set()
- MQMessage
  Excluding:
  - groupId
  - messageFlags
  - messageSequenceNumber
  - offset
  - originalLength
- MQPoolServices
- MQPoolServicesEvent
- MQPoolServicesEventListener
- MQPoolToken
- MQPutMessageOptions
  Excluding:
  - knownDestCount
  - unknownDestCount
  - invalidDestCount
  - recordFields
- MQProcess
- MQQueue
- MQQueueManager
Core details

Excluding:
- begin()
- accessDistributionList()

- MQSimpleConnectionManager
- MQC

Notes:
1. Some constants are not included in the core (see "Restrictions and variations for core classes" for details); do not use them in completely portable programs.
2. Some platforms do not support all connection modes. On these platforms, you can use only the core classes and options that relate to the supported modes. (See Table 1 on page 5)

Restrictions and variations for core classes

The core classes generally behave consistently across all environments, even if the equivalent MQI calls normally have environment differences. The behavior is as if a Windows or UNIX WebSphere MQ queue manager is used, except for the following minor restrictions and variations.

MQGMO_* values

The following MQGMO_* values are not supported by all queue managers, and their use might throw MQException from an MQQueue.get():
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_LOCK
- MQGMO_UNLOCK
- MQGMO_LOGICAL_ORDER
- MQGMO_COMPLETE_MESSAGE
- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE

Additionally, MQGMO_SET_SIGNAL is not supported when used from Java.

MQPMRF_* values

These are used only when putting messages to a distribution list, and are supported only by queue managers supporting distribution lists. For example, z/OS and OS/390 queue managers do not support distribution lists.

MQPMO_* values

The following MQPMO_* values are not supported by all queue managers, and their use might throw MQException from an MQQueue.put() or an MQQueueManager.put():
- MQPMO_NEW_MESSAGE_ID
- MQPMO_NEW_CORREL_ID
- MQPMO_LOGICAL_ORDER

MQCNO_FASTPATH_BINDING

This value is ignored on queue managers that do not support it, or when using a TCP/IP client connection.
MQRO_* values

The following report options can be set but are ignored by some queue managers. This can affect applications connected to a queue manager that honors the report options when the report message is generated by a remote queue manager that does not. Avoid relying on these options if there is a possibility that a queue manager involved does not support them.

- MQRO_EXCEPTION_WITH_FULL_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_COA_WITH_FULL_DATA
- MQRO_COD_WITH_FULL_DATA
- MQRO_DISCARD_MSG

Miscellaneous differences with z/OS and OS/390

Message priority

When a message is put with a priority greater than MaxPriority, a z/OS or OS/390 queue manager rejects the put with MQCC_FAILED and MQRC_PRIORITY_ERROR. Other platforms complete the put with MQCC_WARNING and MQRC_PRIORITY_EXCEEDS_MAXIMUM, and treat the message as if it were put with MaxPriority.

BackoutCount

A z/OS or OS/390 queue manager returns a maximum BackoutCount of 255, even if the message has been backed out more than 255 times.

Default dynamic queue prefix

When connected to a z/OS or OS/390 queue manager using a bindings connection, the default dynamic queue prefix is CSQ.*. Otherwise, the default dynamic queue prefix is AMQ.*.

MQQueueManager constructor

Client connect is not supported on z/OS and OS/390. Attempting to connect with client options results in an MQException with MQCC_FAILED and MQRC_ENVIRONMENT_ERROR. The MQQueueManager constructor might also fail with MQRC_CHAR_CONVERSION_ERROR (if it fails to initialize conversion between the IBM-1047 and ISO8859-1 code pages), or MQRC_UCS2_CONVERSION_ERROR (if it fails to initialize conversion between the queue manager’s code page and Unicode). If your application fails with one of these reason codes, ensure that the National Language Resources component of Language Environment® is installed, and ensure that the correct conversion tables are available.

Conversion tables for Unicode are installed as part of the OS/390 C/C++ optional feature. See the OS/390: C/C++ Programming Guide (SC09-2362) for more information about enabling UCS-2 conversions.
Features outside the core

The WebSphere MQ classes for Java contain the following functions that are specifically designed to use API extensions that are not supported by all queue managers. This section describes how they behave when using a queue manager that does not support them.

**MQQueueManager constructor option**

The MQQueueManager constructor includes an optional integer argument. This maps onto the MQI’s MQCNO options field, and is used to switch between normal and fast path connection. This extended form of the constructor is accepted in all environments, provided that the only options used are MQCNO_STANDARD_BINDING or MQCNO_FASTPATH_BINDING. Any other options cause the constructor to fail with MQRC_OPTIONS_ERROR. The fast path option MQC.MQCNO_FASTPATH_BINDING is honored only when with a bindings connection to a queue manager that supports it. In other environments, it is ignored.

**MQQueueManager.begin() method**

This can be used only against a WebSphere MQ queue manager on UNIX or Windows systems in bindings mode. Otherwise, it fails with MQRC_ENVIRONMENT_ERROR. See “JTA/JDBC coordination using WebSphere MQ base Java” on page 87 for more details.

**MQGetMessageOptions fields**

When using a queue manager that does not support the Version 2 MQGMO structure, leave the following fields set to their default values:

- GroupStatus
- SegmentStatus
- Segmentation

Also, the MatchOptions field support only MQMO_MATCH_MSG_ID and MQMO_MATCH_CORREL_ID. If you put unsupported values into these fields, the subsequent MQQueue.get() fail with MQRC_GMO_ERROR. If the queue manager does not support the Version 2 MQGMO structure, these fields are not updated after a successful MQQueue.get().

**Distribution lists**

The following classes are used to create distribution lists:

- MQDistributionList
- MQDistributionListItem
- MQMessageTracker

You can create and populate MQDistributionLists and MQDistributionListItems in any environment, but not all queue managers allow you to open an MQDistributionList. In particular, z/OS and OS/390 queue managers do not support distribution lists. Attempting to open an MQDistributionList when using such a queue manager results in MQRC_OD_ERROR.

**MQPutMessageOptions fields**

Four fields in the MQPMO are rendered as the following member variables in the MQPutMessageOptions class:

- knownDestCount
unknownDestCount
invalidDestCount
recordFields

These fields are primarily intended for use with distribution lists. However, a queue manager that supports distribution lists also fills in the DestCount fields after an MQPUT to a single queue. For example, if the queue resolves to a local queue, knownDestCount is set to 1 and the other two count fields are set to 0.

If the queue manager does not support distribution lists, these values are simulated as follows:
- If the put() succeeds, unknownDestCount is set to 1, and the others are set to 0.
- If the put() fails, invalidDestCount is set to 1, and the others are set to 0.

The recordFields variable is used with distribution lists. A value can be written into recordFields at any time, regardless of the environment. It is ignored if the MQPutMessageOptions object is used on a subsequent MQQueue.put() or MQQueueManager.put(), rather than MQDistributionList.put().

**MQMD fields**

The following MQMD fields are largely concerned with message segmentation:
- GroupId
- MsgSeqNumber
- Offset
- MsgFlags
- OriginalLength

If an application sets any of these MQMD fields to values other than their defaults, and then does a put() or get() on a queue manager that does not support these, the put() or get() raises an MQException with MQRC_MD_ERROR. A successful put() or get() with such a queue manager always leaves the MQMD fields set to their default values. Do not send a grouped or segmented message to a Java application that runs against a queue manager that does not support message grouping and segmentation.

If a Java application attempts to get() a message from a queue manager that does not support these fields, and the physical message to be retrieved is part of a group of segmented messages (that is, it has non-default values for the MQMD fields), it is retrieved without error. However, the MQMD fields in the MQMessage are not updated, the MQMessage format property is set to MQFMT_MD_EXTENSION, and the true message data is prefixed with an MQMDE structure that contains the values for the new fields.
Chapter 9. The WebSphere MQ base Java classes and interfaces

This chapter describes all the WebSphere MQ classes for Java classes and interfaces. It includes details of the variables, constructors, and methods in each class and interface.

The following classes are described:
- MQChannelDefinition
- MQChannelExit
- MQDistributionList
- MQDistributionListItem
- MQEnvironment
- MQException
- MQGetMessageOptions
- MQManagedObject
- MQMessage
- MQMessageTracker
- MQPoolServices
- MQPoolServicesEvent
- MQPoolToken
- MQPutMessageOptions
- MQProcess
- MQQueue
- MQQueueManager
- MQSimpleConnectionManager

The following interfaces are described:
- MQC
- MQPoolServicesEventListener
- MQConnectionManager
- MQReceiveExit
- MQSecurityExit
- MQSendExit
- ManagedConnection
- ManagedConnectionFactory
- ManagedConnectionMetaData
MQChannelDefinition

public class MQChannelDefinition extends Object

Use the MQChannelDefinition class to pass information concerning the connection to the queue manager to the send, receive, and security exits.

Note: This class does not apply when connecting directly to WebSphere MQ in bindings mode.

Variables

channelName
   public String channelName
   The name of the channel through which the connection is established.

connectionName
   public String connectionName
   The TCP/IP hostname of the machine on which the queue manager resides.

maxMessageLength
   public int maxMessageLength
   The maximum length of message that can be sent to the queue manager.

queueManagerName
   public String queueManagerName
   The name of the queue manager to which the connection is made.

receiveUserData
   public String receiveUserData
   A storage area for the receive exit to use. Information placed here is preserved across invocations of the receive exit, and is also available to the send and security exits.

remotePassword
   public String remotePassword
   The password used to establish the connection.

remoteUserId
   public String remoteUserId
   The user id used to establish the connection.

securityUserData
   public String securityUserData
   A storage area for the security exit to use. Information placed here is preserved across invocations of the security exit, and is also available to the send and receive exits.

sendUserData
   public String sendUserData
A storage area for the send exit to use. Information placed here is preserved across invocations of the send exit, and is also available to the security and receive exits.

**sslPeerName**

```
public String sslPeerName
```

If SSL is used to encrypt data on the wire, this is set to the Distinguished Name presented by the queue manager during connection. If SSL is not used, it is left at null.

**Constructors**

```
MQChannelDefinition
public MQChannelDefinition()
```
public class MQChannelExit
extends Object

This class defines context information passed to the send, receive, and security exits when they are invoked. The exit must set the exitResponse member variable to indicate what action the WebSphere MQ Client for Java should take next.

Note: This class does not apply when connecting directly to WebSphere MQ in bindings mode.

Variables

MQXCC_CLOSE_CHANNEL
    public final static int MQXCC_CLOSE_CHANNEL
MQXCC_OK
    public final static int MQXCC_OK
MQXCC_SUPPRESS_FUNCTION
    public final static int MQXCC_SUPPRESS_FUNCTION
MQXCC_SEND_AND_REQUEST_SEC_MSG
    public final static int MQXCC_SEND_AND_REQUEST_SEC_MSG
MQXCC_SEND_SEC_MSG
    public final static int MQXCC_SEND_SEC_MSG
MQXCC SUPPRESS_EXIT
    public final static int MQXCC_SUPPRESS_EXIT
MQXR_INIT
    public final static int MQXR_INIT
MQXR_INIT_SEC
    public final static int MQXR_INIT_SEC
MQXR_SEC_MSG
    public final static int MQXR_SEC_MSG
MQXR_TERM
    public final static int MQXR_TERM
MQXR_XMIT
    public final static int MQXR_XMIT
MQXT_CHANNEL_SEC_EXIT
    public final static int MQXT_CHANNEL_SEC_EXIT
MQXT_CHANNEL_SEND_EXIT
    public final static int MQXT_CHANNEL_SEND_EXIT
MQXT_CHANNEL_RCV_EXIT
    public final static int MQXT_CHANNEL_RCV_EXIT

capabilityFlags
    public static final int capabilityFlags
Indicates the capability of the queue manager.

Only the MQC.MQCF_DIST_LISTS flag is supported.

**exitID**

`public int exitID`

The type of exit that has been invoked. For an MQSecurityExit this is always MQXT_CHANNEL_SEC_EXIT; for an MQSendExit this is always MQXT_CHANNEL_SEND_EXIT; for an MQReceiveExit this is always MQXT_CHANNEL_RCV_EXIT.

**exitReason**

`public int exitReason`

The reason for invoking the exit. Possible values are:

**MQXR_INIT**

Exit initialization; called after the channel connection conditions have been negotiated, but before any security flows have been sent.

**MQXR_INIT_SEC**

Indicates that the exit is to initiate the security dialog with the queue manager.

**MQXR_SEC_MSG**

Indicates to the security exit that a security message has been received from the queue manager.

**MQXR_TERM**

Exit termination; called after the disconnect flows have been sent but before the socket connection is destroyed.

**MQXR_XMIT**

For a send exit, indicates that data is to be transmitted to the queue manager.

For a receive exit, indicates that data has been received from the queue manager.

**exitResponse**

`public int exitResponse`

Set by the exit to indicate the action that WebSphere MQ classes for Java should take next. Valid values are:

**MQXCC_CLOSE_CHANNEL**

Set by any exit to indicate that the connection to the queue manager should be closed.

**MQXCC_OK**

Set by the security exit to indicate that security exchanges are complete.

Set by send exit to indicate that the returned data is to be transmitted to the queue manager.

Set by the receive exit to indicate that the returned data is available for processing by the WebSphere MQ Client for Java.

**MQXCC_SEND_AND_REQUEST_SEC_MSG**

Set by the security exit to indicate that the returned data is to be transmitted to the queue manager, and that a response is expected from the queue manager.
MQChannelExit

**MQXCC_SEND_SEC_MSG**
Set by the security exit to indicate that the returned data is to be transmitted to the queue manager, and that no response is expected.

**MQXCC_SUPPRESS_EXIT**
Set by any exit to indicate that it should no longer be called.

**MQXCC_SUPPRESS_FUNCTION**
Set by the security exit to indicate that communications with the queue manager should be shut down.

```java
exitUserArea
public byte exitUserArea[]
A storage area available for the exit to use.
Any data placed in the exitUserArea is preserved by the WebSphere MQ Client for Java across exit invocations with the same exitID. (That is, the send, receive, and security exits each have their own, independent, user areas.)
```

```java
fapLevel
public static final int fapLevel
The negotiated Format and Protocol (FAP) level.
```

```java
maxSegmentLength
public int maxSegmentLength
The maximum length for any one transmission to a queue manager.
If the exit returns data that is to be sent to the queue manager, the length of the returned data must not exceed this value.
```

**Constructors**

```java
MQChannelExit
public MQChannelExit()
```
MQDistributionList

Create an MQDistributionList using the MQDistributionList constructor or the accessDistributionList method for MQQueueManager.

A distribution list represents a set of open queues to which messages can be sent using a single call to the put() method. (See "Distribution lists" in the WebSphere MQ Application Programming Guide)

Constructors

MQDistributionList

public class MQDistributionList
extends MQManagedObject (See page 123)

qMgr is the queue manager where the list is to be opened.
litems are the items to be included in the distribution list.

See "accessDistributionList" on page 166 for details of the remaining parameters.

Methods

getFirstDistributionListItem

public MQDistributionListItem getFirstDistributionListItem()

Returns the first item in the distribution list, or null if the list is empty.

getInvalidDestinationCount

public int getInvalidDestinationCount()

Returns the number of items in the distribution list that failed to open successfully.

getValidDestinationCount

public int getValidDestinationCount()

Returns the number of items in the distribution list that were opened successfully.

put

public synchronized void put(MQMessage message,
MQPutMessageOptions putMessageOptions )
throws MQException

Puts a message to the queues on the distribution list.
Parameters

message
An input/output parameter containing the message descriptor information and the returned message data.

putMessageOptions
Options that control the action of MQPUT. (See "MQPutMessageOptions" on page 152 for details.)

Throws MQException if the put fails.
MQDistributionListItem

java.lang.Object
  └─ com.ibm.mq.MQMessageTracker
      └─ com.ibm.mq.MQDistributionListItem

public class MQDistributionListItem
  extends MQMessageTracker (See page 144)

An MQDistributionListItem represents a single item (queue) within a distribution list.

Variables

completionCode
  public int completionCode

  The completion code resulting from the last operation on this item. If this was the construction of an MQDistributionList, the completion code relates to the opening of the queue. If it was a put operation, the completion code relates to the attempt to put a message onto this queue.

  The initial value is 0.

queueManagerName
  public String queueManagerName

  The name of the queue manager on which the queue is defined.

  The initial value is "".

queueName
  public String queueName

  The name of a queue you want to use with a distribution list. This cannot be the name of a model queue.

  The initial value is "".

reasonCode
  public int reasonCode

  The reason code resulting from the last operation on this item. If this was the construction of an MQDistributionList, the reason code relates to the opening of the queue. If it was a put operation, the reason code relates to the attempt to put a message onto this queue.

  The initial value is 0.

Constructors

MQDistributionListItem
  public MQDistributionListItem()

  Construct a new MQDistributionListItem object.
public class MQEnvironment
extends Object

Note: All the methods and attributes of this class apply to the WebSphere MQ classes for Java client connections, but only enableTracing, disableTracing, properties, and version_notice apply to bindings connections.

MQEnvironment contains static member variables that control the environment in which an MQQueueManager object (and its corresponding connection to WebSphere MQ) is constructed.

Values set in the MQEnvironment class take effect when the MQQueueManager constructor is called, so set the values in the MQEnvironment class before you construct an MQQueueManager instance.

Variables

Note: Variables marked with * do not apply when connecting directly to WebSphere MQ in bindings mode.

CCSID*

public static int CCSID

The CCSID used by the client.

Changing this value affects the way that the queue manager you connect to translates information in the WebSphere MQ headers. All data in WebSphere MQ headers is drawn from the invariant part of the ASCII codeset, except for the data in the applicationIdData and the putApplicationName fields of the MQMessage class. (See "MQMessage" on page 126.)

If you avoid using characters from the variant part of the ASCII codeset for these two fields, you are then safe to change the CCSID from 819 to any other ASCII codeset.

If you change the client’s CCSID to be the same as that of the queue manager to which you are connecting, you gain a performance benefit at the queue manager because it does not attempt to translate the message headers.

The default value is 819.

channel*

public static String channel

The name of the channel to connect to on the target queue manager. You must set this member variable, or the corresponding property, before constructing an MQQueueManager instance for use in client mode.
hostname*

public static String hostname

The TCP/IP hostname of the machine on which the WebSphere MQ server resides. If the hostname is not set, and no overriding properties are set, bindings mode is used to connect to the local queue manager.

localAddressSetting*

public static String localAddressSetting

The local address, including a range of ports, that is used when connecting to a WebSphere MQ queue manager through a firewall. The format of a local address is [ip-addr][low-port-high-port]). Here are some examples:

9.20.4.98

The channel binds to address 9.20.4.98 locally

9.20.4.98(1000)

The channel binds to address 9.20.4.98 locally and uses port 1000

9.20.4.98(1000,2000)

The channel binds to address 9.20.4.98 locally and uses a port in the range 1000 to 2000

(1000) The channel binds to port 1000 locally

(1000,2000)

The channel binds to a port in the range 1000 to 2000 locally

You can specify a host name instead of an IP address. The variable is initialized from a system property called com.ibm.mq.localAddress when you start the JVM. The default value is null.

password*

public static String password

Equivalent to the WebSphere MQ environment variable MQ_PASSWORD.

If a security exit is not defined for this client, the value of password is transmitted to the server and is available to the server security exit when it is invoked. Use the value to verify the identity of the WebSphere MQ client.

The default value is "".

port*

public static int port

The port to connect to. This is the port on which the WebSphere MQ server is listening for incoming connection requests. The default value is 1414.

properties

public static java.util.Hashtable properties

A set of key/value pairs defining the WebSphere MQ environment. This hash table allows you to set environment properties as key/value pairs rather than as individual variables.

The properties can also be passed as a Hashtable in a parameter on the MQQueueManager constructor. Properties passed on the constructor take precedence over values set with this properties variable, but they are otherwise interchangeable. The order of precedence of finding properties is:

1. properties parameter on MQQueueManager constructor
2. MQEnvironment.properties
3. Other MQEnvironment variables
4. Constant default values

The possible key/value pairs are shown in the following table:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQC.CCSID_PROPERTY</td>
<td>Integer (overrides MQEnvironment.CCSID)</td>
</tr>
<tr>
<td>MQC.CHANNEL_PROPERTY</td>
<td>String (overrides MQEnvironment.channel)</td>
</tr>
<tr>
<td>MQC.CONNECT_OPTIONSPROPERTY</td>
<td>Integer, defaults to MQC.MQCNO_NONE</td>
</tr>
<tr>
<td>MQC.HOST_NAMEPROPERTY</td>
<td>String (overrides MQEnvironment.hostname)</td>
</tr>
<tr>
<td>MQC.LOCAL_ADDRESS_PROPERTY</td>
<td>String (overrides MQEnvironment.localAddressSetting)</td>
</tr>
<tr>
<td>MQC.ORBPROPERTY</td>
<td>org.omg.CORBA.ORB (optional)</td>
</tr>
<tr>
<td>MQC.PASSWORDPROPERTY</td>
<td>String (overrides MQEnvironment.password)</td>
</tr>
<tr>
<td>MQC.PORT_PROPERTY</td>
<td>Integer (overrides MQEnvironment.port)</td>
</tr>
<tr>
<td>MQC.RECEIVE_EXITPROPERTY</td>
<td>MQReceiveExit (overrides MQEnvironment.receiveExit)</td>
</tr>
<tr>
<td>MQC.SECURITY_EXITPROPERTY</td>
<td>MQSecurityExit (overrides MQEnvironment.securityExit)</td>
</tr>
<tr>
<td>MQC.SEND_EXITPROPERTY</td>
<td>MQSendExit (overrides MQEnvironment.sendExit)</td>
</tr>
<tr>
<td>MQC.SSL_CERT_STOREPROPERTY</td>
<td>java.util.Collection, or java.security.cert.CertStore (overrides MQEnvironment.sslCertStores)</td>
</tr>
<tr>
<td>MQC.SSL_CIPHER_SUITEPROPERTY</td>
<td>String (overrides MQEnvironment.sslCipherSuite)</td>
</tr>
<tr>
<td>MQC.SSL_PEER_NAMEPROPERTY</td>
<td>String (overrides MQEnvironment.sslPeerName)</td>
</tr>
<tr>
<td>MQC.SSL_SOCKET_FACTORY_PROPERTY</td>
<td>javax.net.ssl.SSLSocketFactory (overrides MQEnvironment.sslSocketFactory)</td>
</tr>
<tr>
<td>MQC.TRANSPORTPROPERTY</td>
<td>MQC.TRANSPORT_MQSERIES_BINDINGS or MQC.TRANSPORT_MQSERIES_CLIENT or MQC.TRANSPORT_MQSERIES (the default, which selects bindings or client, based on the value of hostname.)</td>
</tr>
<tr>
<td>MQC.USER_IDPROPERTY</td>
<td>String (overrides MQEnvironment.userID.)</td>
</tr>
</tbody>
</table>

receiveExit*

```java
public static MQReceiveExit receiveExit
```

A receive exit allows you to examine, and possibly alter, data received from a queue manager. It is normally used in conjunction with a corresponding send exit at the queue manager.

To provide your own receive exit, define a class that implements the MQReceiveExit interface, and assign receiveExit to an instance of that class. Otherwise, you can leave receiveExit set to null, in which case no receive exit is called.

See also “MQReceiveExit” on page 182
securityExit*

`public static MQSecurityExit securityExit`

A security exit allows you to customize the security flows that occur when an attempt is made to connect to a queue manager.

To provide your own security exit, define a class that implements the MQSecurityExit interface, and assign securityExit to an instance of that class. Otherwise, you can leave securityExit set to null, in which case no security exit is called.

See also [“MQSecurityExit” on page 184](#).

sendExit*

`public static MQSendExit sendExit`

A send exit allows you to examine, and possibly alter, the data sent to a queue manager. It is normally used in conjunction with a corresponding receive exit at the queue manager.

To provide your own send exit, define a class that implements the MQSendExit interface, and assign sendExit to an instance of that class. Otherwise, you can leave sendExit set to null, in which case no send exit is called.

See also [“MQSendExit” on page 186](#).

sslCertStores*

`public static java.util.Collection sslCertStores`

A Collection of CertStore objects used for certificate revocation checking. Use of this variable requires a JVM at Java 2 v1.4 or later. If sslCipherSuite is set, this variable can be used to ensure that the queue manager’s certificate has not become revoked. Each CertStore in the Collection represents an identical copy of the certificate revocation list (CRL). For more information on the behaviour of sslCertStores, refer to [“Using certificate revocation lists” on page 91](#). If set to null (default), the certificate presented by the queue manager is not checked against any certificate revocation list. This variable is ignored if sslCipherSuite is null.

sslCipherSuite*

`public static String sslCipherSuite`

If set, SSL is enabled for the connection. Set the sslCipherSuite to the CipherSuite name matching the CipherSpec set on the SVRCONN channel. If set to null (default), no SSL encryption is performed.

sslPeerName*

`public static String sslPeerName`

A distinguished name pattern. If sslCipherSuite is set, this variable can be used to ensure the correct queue manager is used. For a description of the format for this value, see [“Using the distinguished name of the queue manager” on page 90](#). If set to null (default), no checking of the queue manager’s DN is performed. This variable is ignored if sslCipherSuite is null.
sslSocketFactory*

`public static javax.net.ssl.SSLSocketFactory sslSocketFactory`

The factory to use when connecting with SSL encryption. If sslCipherSuite is set, this variable can be used to customize all aspects of the SSL connection. For more information on constructing and customizing SSLSocketFactory instances, refer to your JSSE provider; for information regarding the use of this variable, refer to "Supplying a customized SSLSocketFactory" on page 92. If set to null (default) and SSL encryption is requested, the default SSLSocketFactory is used. This variable is ignored if sslCipherSuite is null.

userID*

`public static String userID`

Equivalent to the WebSphere MQ environment variable MQ_USER_ID. If a security exit is not defined for this client, the value of userID is transmitted to the server and is available to the server security exit when it is invoked. Use the value to verify the identity of the WebSphere MQ client.

The default value is "".

`version_notice`

`public final static String version_notice`

The current version of WebSphere MQ classes for Java.

Constructors

`MQEnvironment`

`public MQEnvironment()`

Methods

`addConnectionPoolToken`

`public static void addConnectionPoolToken(MQPoolToken token)`

Adds the supplied MQPoolToken to the set of tokens. A default ConnectionManager can use this as a hint; typically, it is enabled only while there is at least one token in the set.

Parameters:

- `token`: The MQPoolToken to add to the set of tokens.

`addConnectionPoolToken`

`public static MQPoolToken addConnectionPoolToken()`

Constructs an MQPoolToken and adds it to the set of tokens. The MQPoolToken is returned to the application to be passed later into removeConnectionPoolToken().
disableTracing
public static void disableTracing()
Turns off the WebSphere MQ Client for Java trace facility.

enableTracing
public static void enableTracing(int level)
Turns on the WebSphere MQ Client for Java trace facility.

Parameters
level The level of tracing required, from 1 to 5 (5 being the most detailed).

enableTracing
public static void enableTracing(int level, OutputStream stream)
Turns on the WebSphere MQ Client for Java trace facility.

Parameters:
level The level of tracing required, from 1 to 5 (5 being the most detailed).
stream The stream to which the trace is written.

getDefaultConnectionManager
public static javax.resource.spi.ConnectionManager getDefaultConnectionManager()
Returns the default ConnectionManager. If the default ConnectionManager is actually an MQConnectionManager, returns null.

getVersionNotice()
public static final String getVersionNotice()
Returns the current version of the WebSphere MQ base Java.

removeConnectionPoolToken
public static void removeConnectionPoolToken(MQPoolToken token)
Removes the specified MQPoolToken from the set of tokens. If that MQPoolToken is not in the set, there is no action.

Parameters:
token The MQPoolToken to remove from the set of tokens.

setDefaultConnectionManager
public static void setDefaultConnectionManager(
MQConnectionManager cxManager)
Sets the supplied MQConnectionManager to be the default ConnectionManager. The default ConnectionManager is used when there is no ConnectionManager specified on the MQQueueManager constructor. This method also empties the set of MQPoolTokens.

Parameters:
cxManager The MQConnectionManager to be the default ConnectionManager.
setDefaultConnectionManager

public static void setDefaultConnectionManager
    (javax.resource.spi.ConnectionManager cxManager)

Sets the default ConnectionManager, and empties the set of
MQPoolTokens. The default ConnectionManager is used when there is no
ConnectionManager specified on the MQQueueManager constructor.

This method requires a JVM at Java 2 v1.3 or later, with JAAS 1.0 or later
installed.

Parameters:

    cxManager
    The default ConnectionManager (which implements the
    javax.resource.spi.ConnectionManager interface).
MQException

java.lang.Object
    └─ java.lang.Throwable
        └─ java.lang.Exception
            └─ com.ibm.mq.MQException

public class MQException
    extends Exception

An MQException is thrown whenever a WebSphere MQ error occurs. You can change the output stream for the exceptions that are logged by setting the value of MQException.log. The default value is System.err. This class contains definitions of completion code and error code constants. Constants beginning MQCC_ are WebSphere MQ completion codes, and constants beginning MQRC_ are WebSphere MQ reason codes. The [WebSphere MQ Application Programming Reference](#) contains a full description of these errors and their probable causes.

Variables

completionCode
    public int completionCode

WebSphere MQ completion code giving rise to the error. The possible values are:
- MQException.MQCC_WARNING
- MQException.MQCC_FAILED

exceptionSource
    public Object exceptionSource

The object instance that threw the exception. You can use this as part of your diagnostics when determining the cause of an error.

log
    public static java.io.OutputStreamWriter log

Stream to which exceptions are logged. (The default is System.err.) If you set this to null, no logging occurs.

reasonCode
    public int reasonCode

WebSphere MQ reason code describing the error. For a full explanation of the reason codes, refer to the [WebSphere MQ Application Programming Reference](#).

Constructors

MQException
    public MQException(int completionCode,
                       int reasonCode,
                       Object source)

Construct a new MQException object.

Parameters
MQException

completionCode
The WebSphere MQ completion code.

reasonCode
The WebSphere MQ reason code.

source  The object in which the error occurred.

Methods

getCode

public Throwable getCause()

Returns the cause of this MQException, or null if the cause is nonexistent or unknown. Note that this method is available on MQException even under JVMs before Java 2 v1.4.

initCause

public Throwable initCause(Throwable cause)

Initializes the cause of this MQException to the specified value. Throws IllegalArgumentException if the cause is this MQException or IllegalStateException if the cause has already been set. Returns this MQException. Note that this method is available on MQException even under JVMs before Java 2 v1.4. Applications do not normally use this method.
MQGetMessageOptions

```java
java.lang.Object
    └─ com.ibm.mq.MQGetMessageOptions

public class MQGetMessageOptions extends Object

This class contains options that control the behavior of MQQueue.get().

Note: The behavior of some of the options available in this class depends on the
environment in which they are used. These elements are marked with a *. See Chapter 8, “Environment-dependent behavior,” on page 95 for details.

Variables

groupStatus*

```java
public char groupStatus
```
This is an output field that indicates whether the retrieved message is in a
group, and if it is, whether it is the last in the group. Possible values are:

MQC.MQGS_LAST_MSG_IN_GROUP
Message is the last in the group. This is also the value returned if
the group consists of only one message.

MQC.MQGS_MSG_IN_GROUP
Message is in a group, but is not the last in the group.

MQC.MQGS_NOT_IN_GROUP
Message is not in a group.

matchOptions*

```java
public int matchOptions
```
Selection criteria that determine which message is retrieved. The following
match options can be set:

MQC.MQMO_MATCH_CORREL_ID
Correlation id to be matched.

MQC.MQMO_MATCH_GROUP_ID
Group id to be matched.

MQC.MQMO_MATCH_MSG_ID
Message id to be matched.

MQC.MQMO_MATCH_MSG_SEQ_NUMBER
Match message sequence number.

MQC.MQMO_NONE
No matching required.

options

```java
public int options
```
Options that control the action of MQQueue.get. Any or none of the
following values can be specified. If more than one option is required, the
values can be added together or combined using the bitwise OR operator.

MQC.MQGMO_ACCEPT_TRUNCATED_MSG
Allow truncation of message data.
MQGetMessageOptions

MQC.MQGMO_BROWSE_FIRST
Browse from start of queue.

MQC.MQGMO_BROWSE_MSG_UNDER_CURSOR*
Browse message under browse cursor.

MQC.MQGMO_BROWSE_NEXT
Browse from the current position in the queue.

MQC.MQGMO_CONVERT
Request the application data to be converted, to conform to the
characterSet and encoding attributes of the MQMessage, before the
data is copied into the message buffer. Because data conversion is
also applied as the data is retrieved from the message buffer,
applications do not usually set this option.

Using this option can cause problems when converting from single
byte character sets to double byte character sets. Instead, do the
conversion using the readString, readLine, and writeString
methods after the message has been delivered.

MQC.MQGMO_FAIL_IF_QUIESCING
Fail if the queue manager is quiescing.

MQC.MQGMO_LOCK*
Lock the message that is browsed.

MQC.MQGMO_MARK_SKIP_BACKOUT*
Allow a unit of work to be backed out without reinstating the
message on the queue.

MQC.MQGMO_MSG_UNDER_CURSOR
Get message under browse cursor.

MQC.MQGMO_NONE
No other options have been specified; all options assume their
default values.

MQC.MQGMO_NO_SYNCPOINT
Get message without syncpoint control.

MQC.MQGMO_NO_WAIT
Return immediately if there is no suitable message.

MQC.MQGMO_SYNCPOINT
Get the message under syncpoint control; the message is marked
as being unavailable to other applications, but it is deleted from
the queue only when the unit of work is committed. The message
is made available again if the unit of work is backed out.

MQC.MQGMO_SYNCPOINT_IF_PERSISTENT*
Get message with syncpoint control if message is persistent.

MQC.MQGMO_UNLOCK*
Unlock a previously locked message.

MQC.MQGMO_WAIT
Wait for a message to arrive.

Segmenting and grouping WebSphere MQ messages can be sent or
received as a single entity, can be split into several segments for sending
and receiving, and can also be linked to other messages in a group.
Each piece of data that is sent is known as a physical message, which can be a complete logical message, or a segment of a longer logical message.

Each physical message usually has a different MsgId. All the segments of a single logical message have the same groupId value and MsgSeqNumber value, but the Offset value is different for each segment. The Offset field gives the offset of the data in the physical message from the start of the logical message. The segments usually have different MsgId values, because they are individual physical messages.

Logical messages that form part of a group have the same groupId value, but each message in the group has a different MsgSeqNumber value. Messages in a group can also be segmented.

The following options can be used for dealing with segmented or grouped messages:

MQC.MQGMO_ALL_MSGS_AVAILABLE*
Retrieve messages from a group only when all the messages in the group are available.

MQC.MQGMO_ALL_SEGMENTS_AVAILABLE*
Retrieve the segments of a logical message only when all the segments in the group are available.

MQC.MQGMO_COMPLETE_MSG*
Retrieve only complete logical messages.

MQC.MQGMO_LOGICAL_ORDER*
Return messages in groups, and segments of logical messages, in logical order.

resolvedQueueName
public String resolvedQueueName
This is an output field that the queue manager sets to the local name of the queue from which the message was retrieved. This is different from the name used to open the queue if an alias queue or model queue was opened.

segmentation*
public char segmentation
This is an output field that indicates whether or not segmentation is allowed for the retrieved message. Possible values are:

MQC.MQSEG_INHIBITED
Segmentation not allowed.

MQC.MQSEG_ALLOWED
Segmentation allowed.

segmentStatus*
public char segmentStatus
This is an output field that indicates whether the retrieved message is a segment of a logical message. If the message is a segment, the flag indicates whether or not it is the last segment. Possible values are:

MQC.MQSS_LAST_SEGMENT
Message is the last segment of the logical message. This is also the value returned if the logical message consists of only one segment.
**MQGetMessageOptions**

**MQC.MQSS_NOT_A_SEGMENT**
Message is not a segment.

**MQC.MQSS_SEGMENT**
Message is a segment, but is not the last segment of the logical message.

**waitInterval**

```java
public int waitInterval
```

The maximum time (in milliseconds) that an MQQueue.get call waits for a suitable message to arrive (used in conjunction with MQC.MQGMO_WAIT). A value of MQC.MQWI_UNLIMITED indicates that an unlimited wait is required.

**Constructors**

**MQGetMessageOptions**

```java
public MQGetMessageOptions()
```

Construct a new MQGetMessageOptions object with options set to MQC.MQGMO_NO_WAIT, a wait interval of zero, and a blank resolved queue name.
MQManagedObject

java.lang.Object
   └─ com.ibm.mq.MQManagedObject

public class MQManagedObject
extends Object

MQManagedObject is a superclass for MQQueueManager, MQQueue, and MQProcess. It provides the ability to inquire and set attributes of these resources.

Variables

alternateUserId

public String alternateUserId

The alternate user ID (if any) specified when this resource was opened. Setting this attribute has no effect.

closeOptions

public int closeOptions

Set this attribute to control the way the resource is closed. The default value is MQC.MQCO_NONE, and this is the only permissible value for all resources other than permanent dynamic queues, and temporary dynamic queues that are being accessed by the objects that created them. For these queues, the following additional values are permissible:

MQC.MQCO_DELETE
   Delete the queue if there are no messages.

MQC.MQCO_DELETE_PURGE
   Delete the queue, purging any messages on it.

connectionReference

public MQQueueManager connectionReference

The queue manager to which this resource belongs. Setting this attribute has no effect.

isOpen

public boolean isOpen

Indicates whether this resource is currently open. This attribute is deprecated and setting it has no effect.

name

public String name

The name of this resource (either the name supplied on the access method, or the name allocated by the queue manager for a dynamic queue). Setting this attribute has no effect.

openOptions

public int openOptions

The options specified when this resource was opened. Setting this attribute has no effect.
MQManagedObject

Constructors

MQManagedObject
protected MQManagedObject()

Constructor method.

Methods

close
public synchronized void close()

Throws MQException.

Closes the object. No further operations against this resource are permitted after this method has been called. To change the behavior of the close method, set the closeOptions attribute.

Throws MQException if the WebSphere MQ call fails.

description
public String getDescription()

Throws MQException.

Returns the description of this resource as held at the queue manager.

If this method is called after the resource has been closed, an MQException is thrown.

inquire
public void inquire(int selectors[], int intAttrs[], byte charAttrs[])

Throws MQException.

Returns an array of integers and a set of character strings containing the attributes of an object (queue, process, or queue manager).

The attributes to be queried are specified in the selectors array. Refer to the [WebSphere MQ Application Programming Reference](#) for details of the permissible selectors and their corresponding integer values.

Many of the more common attributes can be queried using the getXXX() methods defined in MQManagedObject, MQQueue, MQQueueManager, and MQProcess.

Parameters

selectors
 Integer array identifying the attributes with values to be inquired on.

intAttrs
 The array in which the integer attribute values are returned. Integer attribute values are returned in the same order as the integer attribute selectors in the selectors array.
The buffer in which the character attributes are returned, concatenated. Character attributes are returned in the same order as the character attribute selectors in the selectors array. The length of each attribute string is fixed for each attribute.

Throws MQException if the inquire fails.

public boolean isOpen()

Returns the value of the isOpen variable.

public synchronized void set(int selectors[],
int intAttrs[],
byte charAttrs[])

Sets the attributes defined in the selector's vector.

The attributes to be set are specified in the selectors array. Refer to the WebSphere MQ Application Programming Reference for details of the permissible selectors and their corresponding integer values.

Some queue attributes can be set using the setXXX() methods defined in MQQueue.

Parameters

selectors
Integer array identifying the attributes with values to be set.

intAttrs
The array of integer attribute values to be set. These values must be in the same order as the integer attribute selectors in the selectors array.

charAttrs
The buffer in which the character attributes to be set are concatenated. These values must be in the same order as the character attribute selectors in the selectors array. The length of each character attribute is fixed.

Throws MQException if the set fails.
MQMessage

public class MQMessage implements DataInput, DataOutput

MQMessage represents both the message descriptor and the data for a WebSphere MQ message. There is a group of readXXX methods for reading data from a message, and a group of writeXXX methods for writing data into a message. The format of numbers and strings used by these read and write methods can be controlled by the encoding and characterSet member variables. The remaining member variables contain control information that accompanies the application message data when a message travels between sending and receiving applications. The application can set values into the member variable before putting a message to a queue and can read values after retrieving a message from a queue.

Variables

accountingToken
public byte accountingToken[]
Part of the identity context of the message; it allows an application to charge for work done as a result of the message.
The default value is MQC.MQACT_NONE.

applicationIdData
public String applicationIdData
Part of the identity context of the message; it is information that is defined by the application suite, and can be used to provide additional information about the message or its originator.
The default value is "".

applicationOriginData
public String applicationOriginData
Information defined by the application that can be used to provide additional information about the origin of the message.
The default value is "".

backoutCount
public int backoutCount
A count of the number of times the message has previously been returned by an MQQueue.get() call as part of a unit of work, and subsequently backed out.
The default value is zero.

characterSet
public int characterSet
The coded character set identifier of character data in the application message data. The behavior of the readString, readLine, and writeString methods is altered accordingly.
The default value for this field is MQC.MQCCSI_Q_MGR. If the default value is used, CharacterSet 819 (iso-8859-1/latin/ibm819) is assumed. The character set values you can use depend upon the JVM you use. Table 13 shows coded character set identifiers and the characterSet values to use:

Table 13. Character set identifiers

<table>
<thead>
<tr>
<th>characterSet</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>ibm037</td>
</tr>
<tr>
<td>437</td>
<td>ibm437 / PC Original</td>
</tr>
<tr>
<td>500</td>
<td>ibm500</td>
</tr>
<tr>
<td>819</td>
<td>iso-8859-1 / latin1 / ibm819</td>
</tr>
<tr>
<td>1200</td>
<td>Unicode</td>
</tr>
<tr>
<td>1208</td>
<td>UTF-8</td>
</tr>
<tr>
<td>273</td>
<td>ibm273</td>
</tr>
<tr>
<td>277</td>
<td>ibm277</td>
</tr>
<tr>
<td>278</td>
<td>ibm278</td>
</tr>
<tr>
<td>280</td>
<td>ibm280</td>
</tr>
<tr>
<td>284</td>
<td>ibm284</td>
</tr>
<tr>
<td>285</td>
<td>ibm285</td>
</tr>
<tr>
<td>297</td>
<td>ibm297</td>
</tr>
<tr>
<td>420</td>
<td>ibm420</td>
</tr>
<tr>
<td>424</td>
<td>ibm424</td>
</tr>
<tr>
<td>737</td>
<td>ibm737 / PC Greek</td>
</tr>
<tr>
<td>775</td>
<td>ibm775 / PC Baltic</td>
</tr>
<tr>
<td>813</td>
<td>iso-8859-7 / greek / ibm813</td>
</tr>
<tr>
<td>838</td>
<td>ibm838</td>
</tr>
<tr>
<td>850</td>
<td>ibm850 / PC Latin 1</td>
</tr>
<tr>
<td>852</td>
<td>ibm852 / PC Latin 2</td>
</tr>
<tr>
<td>855</td>
<td>ibm855 / PC Cyrillic</td>
</tr>
<tr>
<td>856</td>
<td>ibm856</td>
</tr>
<tr>
<td>857</td>
<td>ibm857 / PC Turkish</td>
</tr>
<tr>
<td>860</td>
<td>ibm860 / PC Portuguese</td>
</tr>
<tr>
<td>861</td>
<td>ibm861 / PC Icelandic</td>
</tr>
<tr>
<td>862</td>
<td>ibm862 / PC Hebrew</td>
</tr>
<tr>
<td>863</td>
<td>ibm863 / PC Canadian French</td>
</tr>
<tr>
<td>864</td>
<td>ibm864 / PC Arabic</td>
</tr>
<tr>
<td>865</td>
<td>ibm865 / PC Nordic</td>
</tr>
<tr>
<td>866</td>
<td>ibm866 / PC Russian</td>
</tr>
<tr>
<td>868</td>
<td>ibm868</td>
</tr>
<tr>
<td>869</td>
<td>ibm869 / PC Modern Greek</td>
</tr>
<tr>
<td>870</td>
<td>ibm870</td>
</tr>
<tr>
<td>871</td>
<td>ibm871</td>
</tr>
<tr>
<td>874</td>
<td>ibm874</td>
</tr>
<tr>
<td>875</td>
<td>ibm875</td>
</tr>
<tr>
<td>912</td>
<td>iso-8859-2 / latin2 / ibm912</td>
</tr>
<tr>
<td>913</td>
<td>iso-8859-3 / latin3 / ibm913</td>
</tr>
<tr>
<td>914</td>
<td>iso-8859-4 / latin4 / ibm914</td>
</tr>
<tr>
<td>915</td>
<td>iso-8859-5 / cyrillic / ibm915</td>
</tr>
<tr>
<td>916</td>
<td>iso-8859-8 / hebrew / ibm916</td>
</tr>
<tr>
<td>918</td>
<td>ibm918</td>
</tr>
<tr>
<td>920</td>
<td>iso-8859-9 / latin5 / ibm920</td>
</tr>
<tr>
<td>921</td>
<td>ibm921</td>
</tr>
<tr>
<td>922</td>
<td>ibm922</td>
</tr>
<tr>
<td>930</td>
<td>ibm930</td>
</tr>
<tr>
<td>932</td>
<td>PC Japanese</td>
</tr>
</tbody>
</table>
Table 13. Character set identifiers (continued)

<table>
<thead>
<tr>
<th>characterSet</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>933</td>
<td>ibm933</td>
</tr>
<tr>
<td>935</td>
<td>ibm935</td>
</tr>
<tr>
<td>937</td>
<td>ibm937</td>
</tr>
<tr>
<td>939</td>
<td>ibm939</td>
</tr>
<tr>
<td>942</td>
<td>ibm942</td>
</tr>
<tr>
<td>948</td>
<td>ibm948</td>
</tr>
<tr>
<td>949</td>
<td>ibm949</td>
</tr>
<tr>
<td>950</td>
<td>ibm950 / Big 5 Traditional Chinese</td>
</tr>
<tr>
<td>954</td>
<td>EUCJIS</td>
</tr>
<tr>
<td>964</td>
<td>ibm964 / CNS 11643 Traditional Chinese</td>
</tr>
<tr>
<td>970</td>
<td>ibm970</td>
</tr>
<tr>
<td>1006</td>
<td>ibm1006</td>
</tr>
<tr>
<td>1025</td>
<td>ibm1025</td>
</tr>
<tr>
<td>1026</td>
<td>ibm1026</td>
</tr>
<tr>
<td>1089</td>
<td>iso-8859-6 / arabic / ibm1089</td>
</tr>
<tr>
<td>1097</td>
<td>ibm1097</td>
</tr>
<tr>
<td>1098</td>
<td>ibm1098</td>
</tr>
<tr>
<td>1112</td>
<td>ibm1112</td>
</tr>
<tr>
<td>1122</td>
<td>ibm1122</td>
</tr>
<tr>
<td>1123</td>
<td>ibm1123</td>
</tr>
<tr>
<td>1124</td>
<td>ibm1124</td>
</tr>
<tr>
<td>1250</td>
<td>Windows Latin 2</td>
</tr>
<tr>
<td>1251</td>
<td>Windows Cyrillic</td>
</tr>
<tr>
<td>1252</td>
<td>Windows Latin 1</td>
</tr>
<tr>
<td>1253</td>
<td>Windows Greek</td>
</tr>
<tr>
<td>1254</td>
<td>Windows Turkish</td>
</tr>
<tr>
<td>1255</td>
<td>Windows Hebrew</td>
</tr>
<tr>
<td>1256</td>
<td>Windows Arabic</td>
</tr>
<tr>
<td>1257</td>
<td>Windows Baltic</td>
</tr>
<tr>
<td>1258</td>
<td>Windows Vietnamese</td>
</tr>
<tr>
<td>1381</td>
<td>ibm1381</td>
</tr>
<tr>
<td>1383</td>
<td>ibm1383</td>
</tr>
<tr>
<td>2022</td>
<td>JIS</td>
</tr>
<tr>
<td>5601</td>
<td>ksc-5601 Korean</td>
</tr>
<tr>
<td>33722</td>
<td>ibm33722</td>
</tr>
</tbody>
</table>

correlationId

    public byte correlationId[]

    For an MQQueue.get() call, the correlation identifier of the message to be retrieved. Normally the queue manager returns the first message with a message identifier and correlation identifier that match those specified. The special value MQC.MQCI_NONE allows any correlation identifier to match.

    For an MQQueue.put() call, this specifies the correlation identifier to use. The default value is MQC.MQCI_NONE.

encoding

    public int encoding
The representation used for numeric values in the application message data; this applies to binary, packed decimal, and floating point data. The behavior of the read and write methods for these numeric formats is altered accordingly.

The following encodings are defined for binary integers:

- **MQC.MQENC_INTEGER_NORMAL**
  - Big-endian integers, as in Java.

- **MQC.MQENC_INTEGER_REVERSED**
  - Little-endian integers, as used by PCs.

The following encodings are defined for packed-decimal integers:

- **MQC.MQENC_DECIMAL_NORMAL**
  - Big-endian packed-decimal, as used by z/OS.

- **MQC.MQENC_DECIMAL_REVERSED**
  - Little-endian packed-decimal.

The following encodings are defined for floating-point numbers:

- **MQC.MQENC_FLOAT_IEEE_NORMAL**
  - Big-endian IEEE floats, as in Java.

- **MQC.MQENC_FLOAT_IEEE_REVERSED**
  - Little-endian IEEE floats, as used by PCs.

- **MQC.MQENC_FLOAT_S390**
  - z/OS format floating points.

Construct a value for the encoding field by adding together one value from each of these three sections (or using the bitwise OR operator). The default value is:

- **MQC.MQENC_INTEGER_NORMAL**  
- **MQC.MQENC_DECIMAL_NORMAL**  
- **MQC.MQENC_FLOAT_IEEE_NORMAL**

For convenience, this value is also represented by **MQC.MQENC_NATIVE**. This setting causes writeInt() to write a big-endian integer, and readInt() to read a big-endian integer. If you set the flag **MQC.MQENC_INTEGER_REVERSED** flag instead, writeInt() writes a little-endian integer, and readInt() reads a little-endian integer.

A loss in precision can occur when converting from IEEE format floating points to zSeries® format floating points.

**expiry**

```java
public int expiry
```

An expiry time expressed in tenths of a second, set by the application that puts the message. After a message’s expiry time has elapsed, it is eligible to be discarded by the queue manager. If the message specified one of the **MQC.MQRO_EXPIRATION** flags, a report is generated when the message is discarded.

The default value is **MQC.MQEI_UNLIMITED**, meaning that the message never expires.

**feedback**

```java
public int feedback
```
MQMessage

Used with a message of type MQC.MQMT_REPORT to indicate the nature of the report. The following feedback codes are defined by the system:

- MQC.MQFB_EXPIRATION
- MQC.MQFB_COA
- MQC.MQFB_COD
- MQC.MQFB.Quit
- MQC.MQFB_PAN
- MQC.MQFB_NAN
- MQC.MQFB_DATA_LENGTH_ZERO
- MQC.MQFB_DATA_LENGTH_NEGATIVE
- MQC.MQFB_DATA_LENGTH_TOO_BIG
- MQC.MQFB_BUFFER_OVERFLOW
- MQC.MQFB_LENGTH_OFF_BY_ONE
- MQC.MQFB_IIF_ERROR

Application-defined feedback values in the range MQC.MQFB_APPL_FIRST to MQC.MQFB_APPL_LAST can also be used.

The default value of this field is MQC.MQFB_NONE, indicating that no feedback is provided.

format

public String format

A format name used by the sender of the message to indicate the nature of the data in the message to the receiver. You can use your own format names, but names beginning with the letters MQ have meanings that are defined by the queue manager. The queue manager built-in formats are:

MQC.MQFMT_ADMIN
  Command server request/reply message.

MQC.MQFMT_COMMAND_1
  Type 1 command reply message.

MQC.MQFMT_COMMAND_2
  Type 2 command reply message.

MQC.MQFMT_DEAD_LETTER_HEADER
  Dead-letter header.

MQC.MQFMT_EVENT
  Event message.

MQC.MQFMT_NONE
  No format name.

MQC.MQFMT_PCF
  User-defined message in programmable command format.

MQC.MQFMT_STRING
  Message consisting entirely of characters.

MQC.MQFMT_TRIGGER
  Trigger message

MQC.MQFMT_XMIT_Q_HEADER
  Transmission queue header.

The default value is MQC.MQFMT_NONE.
groupId
public byte[] groupId
A byte string that identifies the message group to which the physical message belongs.
The default value is MQC.MQGI.NONE.

messageFlags
public int messageFlags
Flags controlling the segmentation and status of a message.

messageId
public byte messageId[]
For an MQQueue.get() call, this field specifies the message identifier of the message to be retrieved. Normally, the queue manager returns the first message with a message identifier and correlation identifier that match those specified. The special value MQC.MQMI_NONE allows any message identifier to match.

For an MQQueue.put() call, this specifies the message identifier to use. If MQC.MQMI_NONE is specified, the queue manager generates a unique message identifier when the message is put. The value of this member variable is updated after the put, to indicate the message identifier that was used.
The default value is MQC.MQMI_NONE.

messageSequenceNumber
public int messageSequenceNumber
The sequence number of a logical message within a group.

messageType
public int messageType
Indicates the type of the message. The following values are currently defined by the system:
- MQC.MQMT_DATAGRAM
- MQC.MQMT_REPLY
- MQC.MQMT_REPORT
- MQC.MQMT_REQUEST

Application-defined values can also be used, in the range MQC.MQMT_APPL_FIRST to MQC.MQMT_APPL_LAST.
The default value of this field is MQC.MQMT_DATAGRAM.

offset
public int offset
In a segmented message, the offset of data in a physical message from the start of a logical message.

originalLength
public int originalLength
The original length of a segmented message.

persistence
public int persistence
Message persistence. The following values are defined:
- MQC.MQPER_NOT_PERSISTENT
**MQMessage**

- MQC.MQPER_PERSISTENT
- MQC.MQPER_PERSISTENCE_AS_Q_DEF

The default value is MQC.MQPER_PERSISTENCE_AS_Q_DEF, which takes the persistence for the message from the default persistence attribute of the destination queue.

**priority**

public int priority

The message priority. The special value MQC.MQPRI_PRIORITY_AS_Q_DEF can also be set in outbound messages, in which case the priority for the message is taken from the default priority attribute of the destination queue.

The default value is MQC.MQPRI_PRIORITY_AS_Q_DEF.

**putApplicationName**

public String putApplicationName

The name of the application that put the message. The default value is "".

**putApplicationType**

public int putApplicationType

The type of application that put the message. This can be a system-defined or user-defined value. The following values are defined by the system:
- MQC.MQAT_AIX
- MQC.MQAT_CICS
- MQC.MQAT_DOS
- MQC.MQAT_IMS
- MQC.MQAT_MVS
- MQC.MQAT_OS2
- MQC.MQAT_OS400
- MQC.MQAT_QMGR
- MQC.MQAT_UNIX
- MQC.MQAT_WINDOWS
- MQC.MQAT_JAVA

The default value is the special value MQC.MQAT_NO_CONTEXT, which indicates that no context information is present in the message.

**putDateTime**

public GregorianCalendar putDateTime

The time and date that the message was put.

**replyToQueueManagerName**

public String replyToQueueManagerName

The name of the queue manager to which reply or report messages should be sent.

The default value is "".

If the value is "" on an MQQueue.put() call, the QueueManager fills in the value.

**replyToQueueName**

public String replyToQueueName
The name of the message queue to which the application that issued the get request for the message should send MQC.MQMT_REPLY and MQC.MQMT_REPORT messages.

The default value is "."

```java
public int report
```

A report is a message about another message. This member variable enables the application sending the original message to specify which report messages are required, whether the application message data is to be included in them, and how to set the message and correlation identifiers in the report or reply. Any, all, or none of the following report types can be requested:

- Exception
- Expiration
- Confirm on arrival
- Confirm on delivery

For each type, only one of the three corresponding values below should be specified, depending on whether the application message data is to be included in the report message.

**Note:** Values marked with ** in the following list are not supported by z/OS queue managers; do not use them if your application is likely to access a z/OS queue manager, regardless of the platform on which the application is running.

The valid values are:

- MQC.MQRO_COA
- MQC.MQRO_COA_WITH_DATA
- MQC.MQRO_COA_WITH_FULL_DATA**
- MQC.MQRO_COD
- MQC.MQRO_COD_WITH_DATA
- MQC.MQRO_COD_WITH_FULL_DATA**
- MQC.MQRO_EXCEPTION
- MQC.MQRO_EXCEPTION_WITH_DATA
- MQC.MQRO_EXCEPTION_WITH_FULL_DATA**
- MQC.MQRO_EXPIRATION
- MQC.MQRO_EXPIRATION_WITH_DATA
- MQC.MQRO_EXPIRATION_WITH_FULL_DATA**

You can specify one of the following to control how the message Id is generated for the report or reply message:

- MQC.MQRO_NEW_MSG_ID
- MQC.MQRO_PASS_MSG_ID

You can specify one of the following to control how the correlation Id of the report or reply message is to be set:

- MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQC.MQRO_PASS_CORREL_ID

You can specify one of the following to control the disposition of the original message when it cannot be delivered to the destination queue:

- MQC.MQRO_DEAD_LETTER_Q
- MQC.MQRO_DISCARD_MSG **
MQMessage

If no report options are specified, the default is:

MQC.MQRO_NEW_MSG_ID | MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID | MQC.MQRO_DEAD_LETTER_Q

You can specify one or both of the following to request that the receiving application sends a positive action or negative action report message.

- MQRO_PAN
- MQRO_NAN

userId  public String userId

Part of the identity context of the message; it identifies the user that originated this message.

The default value is "".

Constructors

MQMessage

public MQMessage()

Creates a new message with default message descriptor information and an empty message buffer.

Methods

clearMessage

public void clearMessage()

Throws IOException.

Discards any data in the message buffer and set the data offset back to zero.

getDataLength

public int getDataLength()

Throws MQException.

The number of bytes of message data remaining to be read.

getDataOffset

public int getDataOffset()

Throws IOException.

Returns the current cursor position within the message data (the point at which read and write operations take effect).

getMessageLength

public int getMessageLength

Throws IOException.

The number of bytes of message data in this MQMessage object.

g getTotalMessageLength

public int getTotalMessageLength()
The total number of bytes in the message as stored on the message queue from which this message was retrieved. When an MQQueue.get() method fails with a message-truncated error code, this method tells you the total size of the message on the queue.

See also "MQQueue.get" on page 156.

getVersion
public int getVersion()
Returns the version of the structure in use.

readBoolean
public boolean readBoolean()

Throws IOException.

Reads a (signed) byte from the current position in the message buffer.

readChar
public char readChar()

Throws IOException, EOFException.

Reads a Unicode character from the current position in the message buffer.

readDecimal2
public short readDecimal2()

Throws IOException, EOFException.

Reads a 2-byte packed decimal number (-999 to 999). The behavior of this method is controlled by the value of the encoding member variable. A value of MQC.MQENC_DECIMAL_NORMAL reads a big-endian packed decimal number; a value of MQC.MQENC_DECIMAL_REVERSED reads a little-endian packed decimal number.

readDecimal4
public int readDecimal4()

Throws IOException, EOFException.

Reads a 4-byte packed decimal number (-9999999 to 9999999). The behavior of this method is controlled by the value of the encoding member variable. A value of MQC.MQENC_DECIMAL_NORMAL reads a big-endian packed decimal number; a value of MQC.MQENC_DECIMAL_REVERSED reads a little-endian packed decimal number.

readDecimal8
public long readDecimal8()

Throws IOException, EOFException.

Reads an 8-byte packed decimal number (-999999999999999 to 999999999999999). The behavior of this method is controlled by the encoding member variable. A value of MQC.MQENC_DECIMAL_NORMAL reads a big-endian packed decimal number; a value of MQC.MQENC_DECIMAL_REVERSED reads a little-endian packed decimal number.
readDouble
public double readDouble()

Throws IOException, EOFException.

Reads a double from the current position in the message buffer. The value of the encoding member variable determines the behavior of this method.

Values of MQC.MQENC_FLOAT_IEEE_NORMAL and MQC.MQENC_FLOAT_IEEE_REVERSED read IEEE standard doubles in big-endian and little-endian formats respectively.

A value of MQC.MQENC_FLOAT_S390 reads a System/390 format floating point number.

readFloat
public float readFloat()

Throws IOException, EOFException.

Reads a float from the current position in the message buffer. The value of the encoding member variable determines the behavior of this method.

Values of MQC.MQENC_FLOAT_IEEE_NORMAL and MQC.MQENC_FLOAT_IEEE_REVERSED read IEEE standard floats in big-endian and little-endian formats respectively.

A value of MQC.MQENC_FLOAT_S390 reads a System/390 format floating point number.

readFully
public void readFully(byte b[])

Throws Exception, EOFException.

Fills the byte array b with data from the message buffer.

readFully
public void readFully(byte b[], int off, int len)

Throws IOException, EOFException.

Fills len elements of the byte array b with data from the message buffer, starting at offset off.

readInt
public int readInt()

Throws IOException, EOFException.

Reads an integer from the current position in the message buffer. The value of the encoding member variable determines the behavior of this method.
A value of MQC.MQENC_INTEGER_NORMAL reads a big-endian integer; a value of MQC.MQENC_INTEGER_REVERSED reads a little-endian integer.

**readInt2**

```java
public short readInt2()
```

Throws IOException, EOFException.

Synonym for readShort(), provided for cross-language WebSphere MQ API compatibility.

**readInt4**

```java
public int readInt4()
```

Throws IOException, EOFException.

Synonym for readInt(), provided for cross-language WebSphere MQ API compatibility.

**readInt8**

```java
public long readInt8()
```

Throws IOException, EOFException.

Synonym for readLong(), provided for cross-language WebSphere MQ API compatibility.

**readLine**

```java
public String readLine()
```

Throws IOException.

Converts from the codeset identified in the CharSet member variable to Unicode, and then reads in a line that has been terminated by \n, \r, \r\n, or EOF.

**readLong**

```java
public long readLong()
```

Throws IOException, EOFException.

Reads a long from the current position in the message buffer. The value of the encoding member variable determines the behavior of this method.

A value of MQC.MQENC_INTEGER_NORMAL reads a big-endian long; a value of MQC.MQENC_INTEGER_REVERSED reads a little-endian long.

**readObject**

```java
public Object readObject()
```

Throws OptionalDataException, ClassNotFoundException, IOException.

Reads an object from the message buffer. The class of the object, the signature of the class, and the value of the non-transient and non-static fields of the class are all read.
public short readShort()

Throws IOException, EOFException.

Reads a short from the current position in the message buffer. The value of the encoding member variable determines the behavior of this method.

A value of MQC.MQENC_INTEGER_NORMAL reads a big-endian short; a value of MQC.MQENC_INTEGER_REVERSED reads a little-endian short.

readString

public String readString(int length)

Throws IOException, EOFException.

Reads a string in the codeset identified by the charset member variable, and convert it into Unicode.

Parameters:

length The number of characters to read (which may differ from the number of bytes according to the codeset, because some codesets use more than one byte per character).

readUInt2

public int readUInt2()

 Throws IOException, EOFException.

Synonym for readUnsignedShort(), provided for cross-language WebSphere MQ API compatibility.

readUnsignedByte

public int readUnsignedByte()

Throws IOException, EOFException.

Reads an unsigned byte from the current position in the message buffer.

readUnsignedShort

public int readUnsignedShort()

Throws IOException, EOFException.

Reads an unsigned short from the current position in the message buffer. The value of the encoding member variable determines the behavior of this method.

A value of MQC.MQENC_INTEGER_NORMAL reads a big-endian unsigned short; a value of MQC.MQENC_INTEGER_REVERSED reads a little-endian unsigned short.

readUTF

public String readUTF()

Throws IOException.
 MQMessage

Reads a UTF string, prefixed by a 2-byte length field, from the current position in the message buffer.

resizeBuffer

```java
public void resizeBuffer(int size)
```

Throws IOException.

A hint to the MQMessage object about the size of buffer that might be required for subsequent get operations. If the message currently contains message data, and the new size is less than the current size, the message data is truncated.

seek

```java
public void seek(int pos)
```

Throws IOException.

Moves the cursor to the absolute position in the message buffer given by pos. Subsequent reads and writes act at this position in the buffer.

Throws EOFException if pos is outside the message data length.

setDataOffset

```java
public void setDataOffset(int offset)
```

Throws IOException.

Moves the cursor to the absolute position in the message buffer. This method is a synonym for seek(), and is provided for cross-language compatibility with the other WebSphere MQ APIs.

setVersion

```java
public void setVersion(int version)
```

Specifies which version of the structure to use. Possible values are:

- MQC.MQMD_VERSION_1
- MQC.MQMD_VERSION_2

You do not need to call this method unless you want to force the client to use a version 1 structure when connected to a queue manager that is capable of handling version 2 structures. In all other situations, the client determines the correct version of the structure to use by querying the queue manager’s capabilities.

skipBytes

```java
public int skipBytes(int n)
```

Throws IOException, EOFException.

Moves forward n bytes in the message buffer.

This method blocks until one of the following occurs:

- All the bytes are skipped
- The end of message buffer is detected
- An exception is thrown

Returns the number of bytes skipped, which is always n.
MQMessage

write
public void write(int b)

Throws IOException.
Writes a byte into the message buffer at the current position.

write
public void write(byte b[])

Throws IOException.
Writes an array of bytes into the message buffer at the current position.

write
public void write(byte b[],
       int off,
       int len)

Throws IOException.
Writes a series of bytes into the message buffer at the current position. len bytes are written, taken from offset off in the array b.

writeBoolean
public void writeBoolean(boolean v)

Throws IOException.
Writes a boolean into the message buffer at the current position.

writeByte
public void writeByte(int v)

Throws IOException.
Writes a byte into the message buffer at the current position.

writeBytes
public void writeBytes(String s)

Throws IOException.
Writes the string to the message buffer as a sequence of bytes. Each character in the string is written in sequence by discarding its high eight bits.

writeChar
public void writeChar(int v)

Throws IOException.
Writes a Unicode character into the message buffer at the current position.

writeChars
public void writeChars(String s)

Throws IOException.
WQMessage

Wante a string as a sequence of Unicode characters into the message buffer at the current position.

**writeDecimal2**

```java
public void writeDecimal2(short v)
```

Throws IOException.

Wante a 2-byte packed decimal format number into the message buffer at the current position. The value of the encoding member variable determines the behavior of this method.

A value of MQC.MQENC_DECIMAL_NORMAL writes a big-endian packed decimal; a value of MQC.MQENC_DECIMAL_REVERSED writes a little-endian packed decimal.

**Parameters**

- `v` can be in the range -999 to 999.

**writeDecimal4**

```java
public void writeDecimal4(int v)
```

Throws IOException.

Wante a 4-byte packed decimal format number into the message buffer at the current position. The value of the encoding member variable determines the behavior of this method.

A value of MQC.MQENC_DECIMAL_NORMAL writes a big-endian packed decimal; a value of MQC.MQENC_DECIMAL_REVERSED writes a little-endian packed decimal.

**Parameters**

- `v` can be in the range -9999999 to 9999999.

**writeDecimal8**

```java
public void writeDecimal8(long v)
```

Throws IOException.

Wante an 8-byte packed decimal format number into the message buffer at the current position. The value of the encoding member variable determines the behavior of this method.

A value of MQC.MQENC_DECIMAL_NORMAL writes a big-endian packed decimal; a value of MQC.MQENC_DECIMAL_REVERSED writes a little-endian packed decimal.

**Parameters:**

- `v` can be in the range -999999999999999 to 999999999999999.

**writeDouble**

```java
public void writeDouble(double v)
```

Throws IOException.
MQMessage

Writes a double into the message buffer at the current position. The value of the encoding member variable determines the behavior of this method.

Values of MQC.MQENC_FLOAT_IEEE_NORMAL and MQC.MQENC_FLOAT_IEEE_REVERSED write IEEE standard floats in big-endian and little-endian formats respectively.

A value of MQC.MQENC_FLOAT_S390 writes a System/390 format floating point number. Note that the range of IEEE doubles is greater than the range of S/390® double precision floating point numbers, so very large numbers cannot be converted.

writeFloat

public void writeFloat(float v)

Throws IOException.

Writes a float into the message buffer at the current position. The value of the encoding member variable determines the behavior of this method.

Values of MQC.MQENC_FLOAT_IEEE_NORMAL and MQC.MQENC_FLOAT_IEEE_REVERSED write IEEE standard floats in big-endian and little-endian formats respectively.

A value of MQC.MQENC_FLOAT_S390 writes a System/390 format floating point number.

writeInt

public void writeInt(int v)

Throws IOException.

Writes an integer into the message buffer at the current position. The value of the encoding member variable determines the behavior of this method.

A value of MQC.MQENC_INTEGER_NORMAL writes a big-endian integer; a value of MQC.MQENC_INTEGER_REVERSED writes a little-endian integer.

writeInt2

public void writeInt2(int v)

Throws IOException.

Synonym for writeShort(), provided for cross-language WebSphere MQ API compatibility.

writeInt4

public void writeInt4(int v)

Throws IOException.

Synonym for writeInt(), provided for cross-language WebSphere MQ API compatibility.

writeInt8

public void writeInt8(long v)
Throws IOException.

Synonym for writeLong(), provided for cross-language WebSphere MQ API compatibility.

**writeLong**

```java
public void writeLong(long v)
```

Throws IOException.

Writes a long into the message buffer at the current position. The value of the encoding member variable determines the behavior of this method.

A value of MQC.MQENC_INTEGER_NORMAL writes a big-endian long; a value of MQC.MQENC_INTEGER_REVERSED writes a little-endian long.

**writeObject**

```java
public void writeObject(Object obj)
```

Throws IOException.

 Writes the specified object to the message buffer. The class of the object, the signature of the class, and the values of the non-transient and non-static fields of the class and all its supertypes are all written.

**writeShort**

```java
public void writeShort(int v)
```

Throws IOException.

Writes a short into the message buffer at the current position. The value of the encoding member variable determines the behavior of this method.

A value of MQC.MQENC_INTEGER_NORMAL writes a big-endian short; a value of MQC.MQENC_INTEGER_REVERSED writes a little-endian short.

**writeString**

```java
public void writeString(String str)
```

Throws IOException.

 Writes a string into the message buffer at the current position, converting it to the codeset identified by the characterSet member variable.

**writeUTF**

```java
public void writeUTF(String str)
```

Throws IOException.

Writes a UTF string, prefixed by a 2-byte length field, into the message buffer at the current position.
MQMessageTracker

MQMessageTracker

public abstract class MQMessageTracker
extends Object

Note: You can use this class only when connected to a WebSphere MQ queue manager.

This class is inherited by MQDistributionListItem (on page 109) where it is used to tailor message parameters for a given destination in a distribution list.

Variables

accountingToken
public byte accountingToken[]

Part of the identity context of the message. It allows an application to charge for work done as a result of the message.

The default value is MQC.MQACT_NONE.

correlationId
public byte correlationId[]

The correlation identifier to use when the message is put.

The default value is MQC.MQCI_NONE.

feedback
public int feedback

Used with a message of type MQC.MQMT_REPORT to indicate the nature of the report. The following feedback codes are defined by the system:
• MQC.MQFB_BUFFER_OVERFLOW
• MQC.MQFB_COA
• MQC.MQFB_COD
• MQC.MQFB_DATA_LENGTH_NEGATIVE
• MQC.MQFB_DATA_LENGTH_TOO_BIG
• MQC.MQFB_DATA_LENGTH_ZERO
• MQC.MQFB_EXPIRATION
• MQC.MQFB_EIIH_ERROR
• MQC.MQFB_LENGTH_OFF_BY_ONE
• MQC.MQFB_NAN
• MQC.MQFB_NONE
• MQC.MQFB_PAN
• MQC.MQFB.Quit

Application-defined feedback values in the range MQC.MQFB_APPL_FIRST to MQC.MQFB_APPL_LAST can also be used.

The default value of this field is MQC.MQFB_NONE, indicating that no feedback is provided.

groupId
public byte[] groupId
A byte string that identifies the message group to which the physical message belongs.

The default value is MQC.MQGI_NONE.

**messageId**

```java
public byte messageId[]
```

The message identifier to use when the message is put. If MQC.MQMI_NONE is specified, the queue manager generates a unique message identifier when the message is put. The value of this member variable is updated after the put to indicate the message identifier that was used.

The default value is MQC.MQMI_NONE.
MQPoolServices

public class MQPoolServices extends Object

Note: Normally, applications do not use this class.

The MQPoolServices class can be used by implementations of ConnectionManager that are intended for use as the default ConnectionManager for WebSphere MQ connections.

A ConnectionManager can construct an MQPoolServices object and, through it, register a listener. This listener receives events that relate to the set of MQPoolTokens that MQEnvironment manages. The ConnectionManager can use this information to perform any necessary startup or cleanup work.

See also “MQPoolServicesEvent” on page 147 and “MQPoolServicesEventListener” on page 180.

Constructors

MQPoolServices

    public MQPoolServices()

    Construct a new MQPoolServices object.

Methods

addMQPoolServicesEventListener

    public void addMQPoolServicesEventListener(MQPoolServicesEventListener listener)

    Adds an MQPoolServicesEventListener. The listener receives an event whenever a token is added or removed from the set of MQPoolTokens that MQEnvironment controls, or whenever the default ConnectionManager changes.

gerTokenCount

    public int getTokenCount()

    Returns the number of MQPoolTokens that are currently registered with MQEnvironment.

removeMQPoolServicesEventListener

    public void removeMQPoolServicesEventListener(MQPoolServicesEventListener listener)

    Removes an MQPoolServicesEventListener.
MQPoolServicesEvent

Note: Normally, applications do not use this class.

An MQPoolServicesEvent is generated whenever an MQPoolToken is added to, or removed from, the set of tokens that MQEnvironment controls. An event is also generated when the default ConnectionManager is changed.

See also "MQPoolServices" on page 146 and "MQPoolServicesEventListener" on page 180.

Variables

**DEFAULT_POOL_CHANGED**

```java
public static final int DEFAULT_POOL_CHANGED
```

The event ID used when the default ConnectionManager changes.

**ID**

```java
protected int ID
```

The event ID. Valid values are:

- `TOKEN_ADDED`
- `TOKEN_REMOVED`
- `DEFAULT_POOL_CHANGED`

**TOKEN_ADDED**

```java
public static final int TOKEN_ADDED
```

The event ID used when an MQPoolToken is added to the set.

**TOKEN_REMOVED**

```java
public static final int TOKEN_REMOVED
```

The event ID used when an MQPoolToken is removed from the set.

**token**

```java
protected MQPoolToken token
```

The token. When the event ID is `DEFAULT_POOL_CHANGED`, this is null.

Constructors

**MQPoolServicesEvent**

```java
public MQPoolServicesEvent(Object source, int eid)
```

Constructs an MQPoolServicesEvent based on the event ID.

**MQPoolServicesEvent**

```java
public MQPoolServicesEvent(Object source, int eid, MQPoolToken token)
```

Constructs an MQPoolServicesEvent based on the event ID and the token.
MQPoolServicesEvent

Methods

**getId**

`public int getId()`

Gets the event ID.

**Returns**

The event ID, with one of the following values:

- DEFAULT_POOL_CHANGED
- TOKEN_ADDED
- TOKEN_REMOVED

**getToken**

`public MQPoolToken getToken()`

Returns the token that was added to, or removed from, the set. If the event ID is DEFAULT_POOL_CHANGED, this is null.
public class MQPoolToken
extends Object

Use an MQPoolToken to enable the default connection pool. MQPoolTokens are registered with the MQEnvironment class before an application component connects to WebSphere MQ. Later, they are deregistered when the component has finished using WebSphere MQ. Typically, the default ConnectionManager is active while the set of registered MQPoolTokens is not empty.

MQPoolToken provides no methods or variables. ConnectionManager providers can choose to extend MQPoolToken so that hints can be passed to the ConnectionManager.

See “MQEnvironment.addConnectionPoolToken” on page 114 and “MQEnvironment.removeConnectionPoolToken” on page 115.

Constructors

MQPoolToken

public MQPoolToken()

Construct a new MQPoolToken object.
MQProcess

public class MQProcess extends MQManagedObject. (See page 123.)

MQProcess provides inquire operations for WebSphere MQ processes.

Constructors

MQProcess

public MQProcess(MQQueueManager qMgr,
            String processName,
            int openOptions,
            String queueManagerName,
            String alternateUserId)
    throws MQException

Access a process on the queue manager qMgr. See accessProcess in the “MQQueueManager” on page 163 for details of the remaining parameters.

Methods

close

public synchronized void close()

Throws MQException.

getApplicationId

public String getApplicationId()

A character string that identifies the application to be started. This information is for use by a trigger monitor application that processes messages on the initiation queue; the information is sent to the initiation queue as part of the trigger message.

Throws MQException if you call this method after you have closed the process.

getApplicationType

public int getApplicationType()

Throws MQException (see page 117).

Identifies the nature of the program to be started in response to the receipt of a trigger message. The application type can take any value, but the following values are recommended for standard types:

- MQC.MQAT_AIX
- MQC.MQAT_CICS
- MQC.MQAT_DOS
- MQC.MQAT_IMS
- MQC.MQAT_MVS
- MQC.MQAT_OS2
- MQC.MQAT_OS400
- MQC.MQAT_UNIX
- MQC.MQAT_WINDOWS
- MQC.MQAT_WINDOWS_NT
- MQC.MWQAT_USER_FIRST (lowest value for user-defined application type)
- MQC.MQAT_USER_LAST (highest value for user-defined application type)

**getEnvironmentData**

```java
public String getEnvironmentData()

throws MQException.
```

A string containing environment-related information pertaining to the application to be started.

**getUserData**

```java
public String getUserData()

throws MQException.
```

A string containing user information relevant to the application to be started.
MQPutMessageOptions

This class contains options that control the behavior of MQQueue.put()..

Note: The behavior of some of the options available in this class depends on the environment in which they are used. These elements are marked with a *.

See Chapter 8, “Environment-dependent behavior,” on page 95 for more details.

Variables

currencyReference

public MQQueue ContextReference

An input field that indicates the source of the context information.

If the options field includes MQC.MQPMO_PASS_IDENTITY_CONTEXT, or MQC.MQPMO_PASS_ALL_CONTEXT, set this field to refer to the MQQueue from which to take the context information.

The initial value of this field is null.

invalidDestCount *

public int invalidDestCount

An output field set by the queue manager to the number of messages that could not be sent to queues in a distribution list. The count includes queues that failed to open as well as queues that were opened successfully, but for which the put operation failed. This field is also set when opening a single queue that is not part of a distribution list.

knownDestCount *

public int knownDestCount

An output field set by the queue manager to the number of messages that the current call has sent successfully to queues that resolve to local queues. This field is also set when opening a single queue that is not part of a distribution list.

options

public int options

Options that control the action of MQQueue.put. Any or none of the following values can be specified. If more than one option is required, the values can be added together or combined using the bitwise OR operator.

MQC.MQPMO_DEFAULT_CONTEXT

Associate default context with the message.

MQC.MQPMO_FAIL_IFQUIESCING

Fail if the queue manager is quiescing.

MQC.MQPMO_LOGICAL_ORDER*

Put logical messages and segments in message groups into their logical order.
MQC.MQPMO_NEW_CORREL_ID*
Generate a new correlation id for each sent message.

MQC.MQPMO_NEW_MSG_ID*
Generate a new message id for each sent message.

MQC.MQPMO_NONE
No options specified. Do not use in conjunction with other options.

MQC.MQPMO_NO_CONTEXT
No context is to be associated with the message.

MQC.MQPMO_NO_SYNCPOINT
Put a message without syncpoint control. Note that, if the syncpoint control option is not specified, a default of no syncpoint is assumed. This applies to all supported platforms.

MQC.MQPMO_PASS_ALL_CONTEXT
Pass all context from an input queue handle.

MQC.MQPMO_PASS_IDENTITY_CONTEXT
Pass identity context from an input queue handle.

MQC.MQPMO_SET_ALL_CONTEXT
Set all context from the application.

MQC.MQPMO_SET_IDENTITY_CONTEXT
Set identity context from the application.

MQC.MQPMO_SYNCPOINT
Put a message with syncpoint control. The message is not visible outside the unit of work until the unit of work is committed. If the unit of work is backed out, the message is deleted.

recordFields *
public int recordFields
Flags indicating which fields are to be customized in each queue when putting a message to a distribution list. One or more of the following flags can be specified:

MQC.MQPMRF_ACCOUNTING_TOKEN
Use the accountingToken attribute in the MQDistributionListItem.

MQC.MQPMRF_CORREL_ID
Use the correlationId attribute in the MQDistributionListItem.

MQC.MQPMRF_FEEDBACK
Use the feedback attribute in the MQDistributionListItem.

MQC.MQPMRF_GROUP_ID
Use the groupld attribute in the MQDistributionListItem.

MQC.MQPMRF_MSG_ID
Use the messageId attribute in the MQDistributionListItem.

The special value MQC.MQPMRF_NONE indicates that no fields are to be customized.

resolvedQueueManagerName
public String resolvedQueueManagerName
An output field set by the queue manager to the name of the queue manager that owns the queue specified by the remote queue name. This
MQPutMessageOptions

might be different from the name of the queue manager from which the queue was accessed if the queue is a remote queue.

resolvedQueueName

public String resolvedQueueName

An output field that is set by the queue manager to the name of the queue on which the message is placed. This might be different from the name used to open the queue if the opened queue was an alias or model queue.

unknownDestCount *

public int unknownDestCount

An output field set by the queue manager to the number of messages that the current call has sent successfully to queues that resolve to remote queues. This field is also set when opening a single queue that is not part of a distribution list.

Constructors

MQPutMessageOptions

public MQPutMessageOptions()

Construct a new MQPutMessageOptions object with no options set, and a blank resolvedQueueName and resolvedQueueManagerName.
MQQueue

java.lang.Object
   │ com.ibm.mq.MQManagedObject
   └─ com.ibm.mq.MQQueue

public class MQQueue extends MQManagedObject. (See page 123)

MQQueue provides inquire, set, put, and get operations for WebSphere MQ queues. The inquire and set capabilities are inherited from MQ.MQManagedObject.

See also “MQQueueManager.accessQueue” on page 168.

Constructors

MQQueue

public MQQueue(MQQueueManager qMgr, String queueName, int openOptions, String queueManagerName, String dynamicQueueName, String alternateUserId )
   throws MQException

Access a queue on the queue manager qMgr.

See “MQQueueManager.accessQueue” on page 168 for details of the remaining parameters.

Methods

close

public synchronized void close()

Throws MQException.

Overrides “MQManagedObject.close” on page 124.

get

public synchronized void get(MQMessage message, MQGetMessageOptions getMessageOptions, int MaxMsgSize)

Throws MQException.

Retrieves a message from the queue, up to a maximum specified message size.

This method takes an MQMessage object as a parameter. It uses some of the fields in the object as input parameters, in particular the messageId and correlationId, so it is important to ensure that these are set as required. (See “Message” on page 349.)

If the get fails, the MQMessage object is unchanged. If it succeeds, the message descriptor (member variables) and message data portions of the MQMessage are completely replaced with the message descriptor and message data from the incoming message.
All calls to WebSphere MQ from a given MQQueueManager are synchronous. Therefore, if you perform a get with wait, all other threads using the same MQQueueManager are blocked from making further WebSphere MQ calls until the get completes. If you need multiple threads to access WebSphere MQ simultaneously, each thread must create its own MQQueueManager object.

Parameters

$message
An input/output parameter containing the message descriptor information and the returned message data.

$getMessageOptions
Options controlling the action of the get. (See "MQGetMessageOptions" on page 119.)

Using option MQC.MQGMO_CONVERT might result in an exception with reason code MQException.MQRC_CONVERTED_STRING_TOO_BIG when converting from single byte character codes to double byte codes. In this case, the message is copied into the buffer but remains encoded using its original character set.

MaxMsgSize
The largest message this call can receive. If the message on the queue is larger than this size, one of two things occurs:

1. If the MQC.MQGMO_ACCEPT_TRUNCATED_MSG flag is set in the options member variable of the MQGetMessageOptions object, the message is filled with as much of the message data as will fit in the specified buffer size, and an exception is thrown with completion code MQException.MQCC_WARNING and reason code MQException.MQRC_TRUNCATED_MSG_ACCEPTED.

2. If the MQC.MQGMO_ACCEPT_TRUNCATED_MSG flag is not set, the message is left on the queue and an MQException is raised with completion code MQException.MQCC_WARNING and reason code MQException.MQRC_TRUNCATED_MSG_FAILED.

Throws MQException if the get fails.

get

public synchronized void get(MQMessage message, MQGetMessageOptions getMessageOptions)

Throws MQException.

Retrieves a message from the queue, regardless of the size of the message. For large messages, the get method might have to issue two calls to WebSphere MQ on your behalf, one to establish the required buffer size and one to get the message data itself.

This method takes an MQMessage object as a parameter. It uses some of the fields in the object as input parameters, in particular the messageId and correlationId, so it is important to ensure that these are set as required. (See "Message" on page 349.)
If the get fails, the MQMessage object is unchanged. If it succeeds, the message descriptor (member variables) and message data portions of the MQMessage are completely replaced with the message descriptor and message data from the incoming message.

All calls to WebSphere MQ from a given MQQueueManager are synchronous. Therefore, if you perform a get with wait, all other threads using the same MQQueueManager are blocked from making further WebSphere MQ calls until the get completes. If you need multiple threads to access WebSphere MQ simultaneously, each thread must create its own MQQueueManager object.

Parameters

message
An input/output parameter containing the message descriptor information and the returned message data.

get(MessageOptions)
Options controlling the action of the get. (See “MQGetMessageOptions” on page 119 for details.)

Throws MQException if the get fails.

get
public synchronized void get(MQMessage message)

A simplified version of the get method previously described.

Parameters

MQMessage
An input/output parameter containing the message descriptor information and the returned message data.

This method uses a default instance of MQGetMessageOptions to do the get. The message option used is MQGMO_NOWAIT.

getCreationDateTime
public GregorianCalendar getCreationDateTime()

Throws MQException.

The date and time that this queue was created.

getQueueType
public int getQueueType()

Throws MQException

Returns

The type of this queue with one of the following values:
- MQC.MQQT_ALIAS
- MQC.MQQT_LOCAL
- MQC.MQQT_REMOTE
- MQC.MQQT_CLUSTER

getCurrentDepth
public int getCurrentDepth()
MQQueue

Throw MQException.

Gets the number of messages currently on the queue. This value is incremented during a put call, and during backout of a get call. It is decremented during a non-browse get and during backout of a put call.

**getDefinitionType**

```java
public int getDefinitionType()
```

Throw MQException.

How the queue was defined.

**Returns**

One of the following:
- MQC.MQQDT_PREDEFINED
- MQC.MQQDT_PERMANENT_DYNAMIC
- MQC.MQQDT_TEMPORARY_DYNAMIC

**getInhibitGet**

```java
public int getInhibitGet()
```

Throw MQException.

Whether get operations are allowed for this queue.

**Returns**

The possible values are:
- MQC.MQQA_GET_INHIBITED
- MQC.MQQA_GET_ALLOWED

**getInhibitPut**

```java
public int getInhibitPut()
```

Throw MQException.

Whether put operations are allowed for this queue.

**Returns**

One of the following:
- MQC.MQQA_PUT_INHIBITED
- MQC.MQQA_PUT_ALLOWED

**getMaximumDepth**

```java
public int getMaximumDepth()
```

Throw MQException.

The maximum number of messages that can exist on the queue at any one time. An attempt to put a message to a queue that already contains this many messages fails with reason code MQException.MQRC_Q_FULL.

**getMaximumMessageLength**

```java
public int getMaximumMessageLength()
```

Throw MQException.
The maximum length of the application data that can exist in each message on this queue. An attempt to put a message larger than this value fails with reason code MQException.MQRC_MSG_TOO_BIG_FOR_Q.

**getOpenInputCount**

```java
public int getOpenInputCount()
```

Throws MQException.

The number of handles that are currently valid for removing messages from the queue. This is the total number of such handles known to the local queue manager, not just those created by the WebSphere MQ classes for Java (using accessQueue).

**getOpenOutputCount**

```java
public int getOpenOutputCount()
```

Throws MQException.

The number of handles that are currently valid for adding messages to the queue. This is the total number of such handles known to the local queue manager, not just those created by the WebSphere MQ classes for Java (using accessQueue).

**getShareability**

```java
public int getShareability()
```

Throws MQException.

Whether the queue can be opened for input multiple times.

**Returns**

One of the following:
- MQC.MQQA_SHAREABLE
- MQC.MQQA_NOT_SHAREABLE

**getTriggerControl**

```java
public int getTriggerControl()
```

Throws MQException.

Whether trigger messages are written to an initiation queue, to start an application to service the queue.

**Returns**

The possible values are:
- MQC.MQT_C_OFF
- MQC.MQT_C_ON

**getTriggerData**

```java
public String getTriggerData()
```

Throws MQException.

The free-format data that the queue manager inserts into the trigger message when a message arriving on this queue causes a trigger message to be written to the initiation queue.

**getTriggerDepth**
public int getTriggerDepth()

Throws MQException.

The number of messages that have to be on the queue before a trigger message is written when trigger type is set to MQC.MQT_DEPTH.

getTriggerMessagePriority

public int getTriggerMessagePriority()

Throws MQException.

The message priority below which messages do not contribute to the generation of trigger messages (that is, the queue manager ignores these messages when deciding whether to generate a trigger). A value of zero causes all messages to contribute to the generation of trigger messages.

getTriggerType

public int getTriggerType()

Throws MQException.

The conditions under which trigger messages are written as a result of messages arriving on this queue.

Returns

The possible values are:

- MQC.MQT_NONE
- MQC.MQT_FIRST
- MQC.MQT_EVERY
- MQC.MQT_DEPTH

put

public synchronized void put(MQMessage message,
MQPutMessageOptions putMessageOptions)

Throws MQException.

Places a message onto the queue.

Note: For simplicity and performance, if you want to put just a single message to a queue, use the put() method on your MQQueueManager object. For this you do not need to have an MQQueue object. See "MQQueueManager.put" on page 174.

This method takes an MQMessage object as a parameter. The message descriptor properties of this object can be altered as a result of this method. The values that they have immediately after the completion of this method are the values that were put onto the WebSphere MQ queue.

Modifications to the MQMessage object after the put has completed do not affect the actual message on the WebSphere MQ queue.

A put updates the messageId and correlationId. Consider this when making further calls to put/get using the same MQMessage object. Also, calling put does not clear the message data, so:
puts two messages. The first contains a and the second ab.

**Parameters**

`message`
Message Buffer containing the Message Descriptor data and message to be sent.

`putMessageOptions`
Options controlling the action of the put. (See “MQPutMessageOptions” on page 152)

Throws MQException if the put fails.

**put**

```java
public synchronized void put(MQMessage message)
```

A simplified version of the put method previously described.

**Parameters**

`MQMessage`
Message Buffer containing the Message Descriptor data and message to be sent.

This method uses a default instance of MQPutMessageOptions to do the put.

**setInhibitGet**

```java
public void setInhibitGet(int inhibit)
```

Throws MQException.

Controls whether get operations are allowed for this queue. The permissible values are:
- MQC.MQQA_GET_INHIBITED
- MQC.MQQA_GET_ALLOWED

**setInhibitPut**

```java
public void setInhibitPut(int inhibit)
```

Throws MQException.

Controls whether put operations are allowed for this queue. The permissible values are:
- MQC.MQQA.PUT_INHIBITED
- MQC.MQQA.PUT_ALLOWED

**setTriggerControl**

```java
public void setTriggerControl(int trigger)
```

Throws MQException.
MQQueue

Controls whether trigger messages are written to an initiation queue to start an application to service the queue. The permissible values are:
- MQC.MQTC_OFF
- MQC.MQTC_ON

**setTriggerData**

```java
public void setTriggerData(String data)
```

Throws MQException.

Sets the free-format data that the queue manager inserts into the trigger message when a message arriving on this queue causes a trigger message to be written to the initiation queue. The maximum permissible length of the string is given by MQC.MQ_TRIGGER_DATA_LENGTH.

**setTriggerDepth**

```java
public void setTriggerDepth(int depth)
```

Throws MQException.

Sets the number of messages that have to be on the queue before a trigger message is written when trigger type is set to MQC.MQTT_DEPTH.

**setTriggerMessagePriority**

```java
public void setTriggerMessagePriority(int priority)
```

Throws MQException.

Sets the message priority below which messages do not contribute to the generation of trigger messages (that is, the queue manager ignores these messages when deciding whether a trigger should be generated). A value of zero causes all messages to contribute to the generation of trigger messages.

**setTriggerType**

```java
public void setTriggerType(int type)
```

Throws MQException.

Sets the conditions under which trigger messages are written as a result of messages arriving on this queue. The possible values are:
- MQC.MQTT_NONE
- MQC.MQTT_FIRST
- MQC.MQTT_EVERY
- MQC.MQTT_DEPTH
MQQueueManager

```
java.lang.Object
    └─ com.ibm.mq.MQManagedObject
        └─ com.ibm.mq.MQQueueManager
```

public class MQQueueManager extends MQManagedObject. (See page 123)

**Note:** The behavior of some of the options available in this class depends on the environment in which they are used. These elements are marked with a *. See Chapter 8, “Environment-dependent behavior,” on page 95 for details.

**Variables**

`isConnected`

```java
public boolean isConnected
```

True if the connection to the queue manager is still open.

**Constructors**

MQQueueManager

```
public MQQueueManager(String queueManagerName)
```

Throws MQException.

Creates a connection to the named queue manager.

**Note:** When using WebSphere MQ classes for Java, the hostname, channel name, and port to use during the connection request are specified in the MQEnvironment class. This must be done before calling this constructor.

The following example shows a connection to a queue manager MYQM, running on a machine with hostname fred.mq.com.

```java
MQEnvironment.hostname = "fred.mq.com"; // host to connect to
MQEnvironment.port     = 1414;          // port to connect to.
                          // If I don't set this,
                          // it defaults to 1414
                          // (the default WebSphere MQ port)
MQEnvironment.channel   = "channel.name"; // the CASE-SENSITIVE
                                      // name of the
                                      // SVR CONN channel on
                                      // the queue manager
MQQueueManager qMgr     = new MQQueueManager("MYQM");
```

If the queue manager name is left blank (null or ""), a connection is made to the default queue manager.

See also "MQEnvironment” on page 110.
public MQQueueManager(String queueManagerName, MQConnectionManager cxManager)

Throws MQException.

Connects to the specified Queue Manager, using the properties in MQEnvironment. The specified MQConnectionManager manages the connection.

public MQQueueManager(String queueManagerName, ConnectionManager cxManager)

Throws MQException.

Connects to the specified Queue Manager, using the properties in MQEnvironment. The specified ConnectionManager manages the connection. If the value of cxManager is null, then the default ConnectionManger is used.

This constructor requires a JVM at at least Java 2 v1.3, with at least JAAS 1.0 installed.

public MQQueueManager(String queueManagerName, int options)

Throws MQException.

This version of the constructor is intended for use only in bindings mode. It uses the extended connection API (MQCONNX) to connect to the queue manager. The options parameter allows you to choose fast or normal bindings. Possible values are:
• MQC.MQCNO_FASTPATH_BINDING for fast bindings *.
• MQC.MQCNO_STANDARD_BINDING for normal bindings.

public MQQueueManager(String queueManagerName, int options, MQConnectionManager cxManager)

Throws MQException.

Performs an MQCONNX, passing the supplied options. The specified MQConnectionManager manages the connection.

public MQQueueManager(String queueManagerName, int options, ConnectionManager cxManager)

Throws MQException.

Performs an MQCONNX, passing the supplied options. The specified ConnectionManager manages the connection.
This constructor requires a JVM at at least Java 2 v1.3, with at least JAAS 1.0 installed.

**MQQueueManager**

```java
public MQQueueManager(String queueManagerName,
                        java.util.Hashtable properties)
```

The properties parameter takes a series of key/value pairs that describe the WebSphere MQ environment for this particular queue manager. These properties, where specified, override the values set by the MQEnvironment class, and allow the individual properties to be set on a queue manager by queue manager basis. See “MQEnvironment.properties” on page 111.

**MQQueueManager**

```java
public MQQueueManager(String queueManagerName,
                        Hashtable properties,
                        MQConnectionManager cxManager)
```

Throws MQException.

This constructor connects to the named Queue Manager, using the supplied hash table of properties to override those in MQEnvironment. The specified MQConnectionManager manages the connection.

**MQQueueManager**

```java
public MQQueueManager(String queueManagerName,
                        Hashtable properties,
                        ConnectionManager cxManager)
```

Throws MQException.

Connects to the named Queue Manager, using the supplied hash table of properties to override those in MQEnvironment. The specified ConnectionManager manages the connection.

This constructor requires a JVM at at least Java 2 v1.3, with at least JAAS 1.0 installed.
Methods

accessDistributionList

`public synchronized MQDistributionList accessDistributionList(MQDistributionListItem[] litems, int openOptions, String alternateUserId)`

Throws MQException.

Parameters

- `litems` The items to be included in the distribution list.
- `openOptions` Options that control the opening of the distribution list.
- `alternateUserId` If MQOO_ALTERNATE_USER_AUTHORITY is specified in the openOptions parameter, specifies the alternate user identifier that is used to check the authorization for the open. If MQOO_ALTERNATE_USER_AUTHORITY is not specified, this parameter can be left blank (or null).

Returns

A newly-created MQDistributionList, which is open and ready for put operations.

Throws MQException if the open fails.

See also “MQQueueManager.accessQueue” on page 168.

accessDistributionList

This is a simplified version of the AccessDistributionList method previously described.

`public synchronized MQDistributionList accessDistributionList(MQDistributionListItem[] litems, int openOptions)`

Throws MQException.

Parameters

- `litems` The items to be included in the distribution list.
- `openOptions` Options that control the opening of the distribution list.
- `alternateUserId` is set to "".

See the full description of “accessDistributionList” above for details of the other parameters.
accessProcess

public synchronized MQProcess accessProcess
    (String processName,
     int openOptions,
     String queueManagerName,
     String alternateUserId)

Throws MQException.

Establishes access to a WebSphere MQ process on this queue manager to inquire about the process attributes.

Parameters

processName
    Name of process to open.

openOptions
    Options that control the opening of the process. Inquire is automatically added to the options specified, so there is no need to specify it explicitly.

Valid options are:

MQC.MQOO_ALTERNATE_USER_AUTHORITY
    Validate with the specified user ID

MQC.MQOO_FAIL_IF_QUIESCING
    Fail if the queue manager is quiescing

If more than one option is required, the values can be added together or combined using the bitwise OR operator. See the WebSphere MQ Application Programming Reference for a fuller description of these options.

queueManagerName
    Name of the queue manager on which the process is defined. Applications should leave this parameter blank or null.

alternateUserId
    If MQOO_ALTERNATE_USER_AUTHORITY is specified in the openOptions parameter, this parameter specifies the alternate user identifier that is used to check the authorization for the open. If MQOO_ALTERNATE_USER_AUTHORITY is not specified, this parameter can be left blank (or null).

accessProcess
    This is a simplified version of the AccessProcess method previously described.

public synchronized MQProcess accessProcess
    (String processName,
     int openOptions)
MQQueueManager

Parameters

processName
   The name of the process to open.

openOptions
   Options that control the opening of the process.

See the full description of “accessProcess” on page 167 above for details of the other options.

queueManagerName and alternateUserId are set to “”.

accessQueue

public synchronized MQQueue accessQueue
   (String queueName, int openOptions,
    String queueManagerName,
    String dynamicQueueName,
    String alternateUserId)

Throws MQException.

Establishes access to a WebSphere MQ queue on this queue manager to get or browse messages, put messages, inquire about the attributes of the queue or set the attributes of the queue.

If the queue named is a model queue, a dynamic local queue is created. The name of the created queue can be determined from the name attribute of the returned MQQueue object.

Parameters

queueName
   Name of queue to open.

openOptions
   Options that control the opening of the queue. Valid options are:

   MQC.MQOO_ALTERNATE_USER_AUTHORITY
      Validate with the specified user identifier.

   MQC.MQOO_BIND_AS_QDEF
      Use default binding for queue.

   MQC.MQOO_BIND_NOT_FIXED
      Do not bind to a specific destination.

   MQC.MQOO_BIND_ON_OPEN
      Bind handle to destination when queue is opened.

   MQC.MQOO_BROWSE
      Open to browse message.

   MQC.MQOO_FAIL_IF QUIESCING
      Fail if the queue manager is quiescing.

   MQC.MQOO_INPUT_AS_Q_DEF
      Open to get messages using queue-defined default.

   MQC.MQOO_INPUT_SHARED
      Open to get messages with shared access.
MQC.MQOO_INPUT_EXCLUSIVE
Open to get messages with exclusive access.

MQC.MQOO_INQUIRE
Open for inquiry - required if you wish to query properties.

MQC.MQOO_OUTPUT
Open to put messages.

MQC.MQOO_PASS_ALL_CONTEXT
Allow all context to be passed.

MQC.MQOO_PASS_IDENTITY_CONTEXT
Allow identity context to be passed.

MQC.MQOO_SAVE_ALL_CONTEXT
Save context when message retrieved*.

MQC.MQOO_SET
Open to set attributes.

MQC.MQOO_SET_ALL_CONTEXT
Allows all context to be set.

MQC.MQOO_SET_IDENTITY_CONTEXT
Allows identity context to be set.

If more than one option is required, the values can be added together or combined using the bitwise OR operator. See the [WebSphere MQ Application Programming Reference](#) for a fuller description of these options.

queueManagerName
Name of the queue manager on which the queue is defined. A name that is entirely blank or null denotes the queue manager to which this MQQueueManager object is connected.

dynamicQueueName
This parameter is ignored unless queueName specifies the name of a model queue. If it does, this parameter specifies the name of the dynamic queue to be created. A blank or null name is not valid if queueName specifies the name of a model queue. If the last non-blank character in the name is an asterisk (*), the queue manager replaces the asterisk with a string of characters that guarantees that the name generated for the queue is unique on this queue manager.

alternateUserId
If MQOO_ALTERNATE_USER_AUTHORITY is specified in the openOptions parameter, this parameter specifies the alternate user identifier that is used to check the authorization for the open. If MQOO_ALTERNATE_USER_AUTHORITY is not specified, this parameter can be left blank (or null).

Returns
MQQueue that has been successfully opened.

Throws MQException if the open fails.
accessQueue

```java
public synchronized MQQueue accessQueue
    (String queueName,
     int openOptions)
```

Throws MQException if you call this method after disconnecting from the queue manager.

**Parameters**

- `queueName` Name of queue to open
- `openOptions` Options that control the opening of the queue

See the description of [“MQQueueManager.accessQueue” on page 168](#) for details of the parameters.

For this version of the method, `queueManagerName`, `dynamicQueueName`, and `alternateUserId` are set to "".

**Returns**

MQProcess that has been successfully opened.

Throws MQException if the open fails.

backout

```java
public synchronized void backout()
```

Throws MQException.

Calling this method indicates to the queue manager that all the message gets and puts that have occurred since the last syncpoint are to be backed out. Messages put as part of a unit of work (with the MQC.MQPMO_SYNCPOINT flag set in the options field of MQPutMessageOptions) are deleted; messages retrieved as part of a unit of work (with the MQC.MQGMO_SYNCPOINT flag set in the options field of MQGetMessageOptions) are reinstated on the queue.

See also the description of the commit method.

begin* (bindings connection only)

```java
public synchronized void begin()
```

Throws MQException.

This method is supported only by the WebSphere MQ classes for Java in bindings mode. It signals to the queue manager that a new unit of work is starting. For a complete description of using this method, see [JTA/JDBC coordination using WebSphere MQ base Java” on page 87](#)

Do not use this method for applications that use local one-phase transactions.
commit

```java
public synchronized void commit()
```

Throws MQException.

Calling this method indicates to the queue manager that the application has reached a syncpoint, and that all the message gets and puts that have occurred since the last syncpoint are to be made permanent. Messages put as part of a unit of work (with the MQC.MQPMO_SYNCPOINT flag set in the options field of MQPutMessageOptions) are made available to other applications. Messages retrieved as part of a unit of work (with the MQC.MQGMO_SYNCPOINT flag set in the options field of MQGetMessageOptions) are deleted.

See also the description of the backout method.

disconnect

```java
public synchronized void disconnect()
```

Throws MQException.

Terminates the connection to the queue manager. All open queues and processes accessed by this queue manager are closed, and become unusable. When you have disconnected from a queue manager, the only way to reconnect is to create a new MQQueueManager object.

Normally, any work performed as part of a unit of work is committed. However, if this connection is managed by a ConnectionManager, rather than an MQConnectionManager, the unit of work might be rolled back.

getcharacterSet

```java
public int getcharacterSet()
```

Throws MQException.

Returns the CCSID (Coded Character Set Identifier) of the queue manager’s codeset. This defines the character set used by the queue manager for all character string fields in the application programming interface.

Throws MQException if you call this method after disconnecting from the queue manager.

gecommandInputQueueName

```java
public String getCommandInputQueueName()
```

Throws MQException.

Returns the name of the command input queue defined on the queue manager. This is a queue to which applications can send commands, if authorized to do so.

Throws MQException if you call this method after disconnecting from the queue manager.
MQQueueManager

getCommandLevel

public int getCommandLevel()

Throws MQException.

Indicates the level of system control commands supported by the queue manager. The set of system control commands that correspond to a particular command level varies according to the architecture of the platform on which the queue manager is running. See the WebSphere MQ documentation for your platform for further details.

Throws MQException if you call this method after disconnecting from the queue manager.

Returns

One of the MQC.MQCMDL_LEVEL_xxx constants

getAddressListCapable

public boolean getDistributionListCapable()

Indicates whether the queue manager supports distribution lists.

getJDBCConnection

public java.sql.Connection getJDBCConnection(XADataSource dataSource, String userid, String password)
throws MQException, SQLException, Exception

Returns a Connection object for use with the JTA-JDBC support.

This method declares Exception in its throws clause to avoid problems with the JVM verifier for customers who are not using the JTA functionality. The actual exception thrown is javax.transaction.xa.XAException, which requires the jta.jar file to be added to the classpath for programs that did not previously require it.

Parameters

dataSource

A database-specific implementation of the XADataSource interface that defines the details of the database to connect to. See the documentation for your database to determine how to create an appropriate XADataSource object to pass into getJDBCConnection.

userid

The user ID to use for this connection to the database. This is passed to the underlying XADataSource.getXAConnection method.

password

The password to use for this connection to the database. This is passed to the underlying XADataSource.getXAConnection method.
getJDBCConnection

public java.sql.Connection getJDBCConnection(javax.sql.XADataSource xads)
    throws MQException, SQLException, Exception

Returns a Connection object for use with the JTA-JDBC support.

This method declares Exception in its throws clause to avoid problems with the JVM verifier for customers who are not using the JTA functionality. The actual exception thrown is javax.transaction.xa.XAException, which requires the jta.jar file to be added to the classpath for programs that did not previously require it.

Parameters

xads  A database-specific implementation of the XADataSource interface that defines the details of the database to connect to. See the documentation for your database to determine how to create an appropriate XADataSource object to pass into getJDBCConnection.

getMaximumMessageLength

public int getMaximumMessageLength()

Throws MQException.

Returns the maximum length of a message (in bytes) that can be handled by the queue manager. No queue can be defined with a maximum message length greater than this.

Throws MQException if you call this method after disconnecting from the queue manager.

getMaximumPriority

public int getMaximumPriority()

Throws MQException.

Returns the maximum message priority supported by the queue manager. Priorities range from zero (lowest) to this value.

Throws MQException if you call this method after disconnecting from the queue manager.

getSyncpointAvailability

public int getSyncpointAvailability()

Throws MQException.

Indicates whether the queue manager supports units of work and syncpointing with the MQQueue.get and MQQueue.put methods.
Returns

- MQC.MQSP_AVAILABLE if syncpointing is available.
- MQC.MQSP_NOT_AVAILABLE if syncpointing is not available.

Throws MQException if you call this method after disconnecting from the queue manager.

isConnected

```java
public boolean isConnected()
```

Returns the value of the isConnected variable.

put

```java
public synchronized void put(String qName,
   String qmName,
   MQMessage msg,
   MQPutMessageOptions pmo,
   String altUserId)
```

Throws MQException.

Places a single message onto a queue without having to create an MQQueue object first.

The qName (queue name) and qmName (queue manager name) parameters identify where the message is placed. If the queue is a model queue, an MQException is thrown.

In other respects, this method behaves like the put method on the MQQueue object. It is an implementation of the MQPUT1 MQI call. See “MQQueue.put” on page 160.

**Parameters**

- **qName** The name of the queue onto which to place the message.
- **qmName** The name of the queue manager on which the queue is defined.
- **msg** The message to send.
- **pmo** Options controlling the actions of the put. See “MQPutMessageOptions” on page 152 for more details.
- **altUserId** Specifies an alternative user identifier used to check authorization when placing the message on a queue. If you do not specify MQPMO_ALTERNATE_USER, this parameter is ignored.

put

```java
public synchronized void put(String qName,
   String qmName,
   MQMessage msg,
   MQPutMessageOptions pmo)
```

Throws MQException.

Places a single message onto a queue without having to create an MQQueue object first.
This version of the method allows you to omit the altUserid parameter. See the fully-specified method ("MQQueueManager.put" on page 174) for details of the parameters.

```java
public synchronized void put(String qName,
       String qmName,
       MQMessage msg)
```

Throws MQException.

Places a single message onto a queue without having to create an MQQueue object first.

This version of the method allows you to omit the put message options (pmo) and altUserid parameters. See the fully-specified method ("MQQueueManager.put" on page 174) for details of the parameters.

```java
public synchronized void put(String qName,
       MQMessage msg,
       MQPutMessageOptions pmo)
```

Throws MQException.

Places a single message onto a queue without having to create an MQQueue object first.

This version of the method allows you to omit the qmName and altUserid parameters. See the fully-specified method ("MQQueueManager.put" on page 174) for details of the parameters.

```java
public synchronized void put(String qName,
       MQMessage msg)
```

Throws MQException.

Places a single message onto a queue without having to create an MQQueue object first.

This version of the method allows you to omit the qmName, put message options (pmo), and altUserid parameters. See the fully-specified method ("MQQueueManager.put" on page 174) for details of the parameters.
MQSimpleConnectionManager

public class MQSimpleConnectionManager implements MQConnectionManager (See page 181.)

An MQSimpleConnectionManager provides basic connection pooling function. You can use an MQSimpleConnectionManager either as the default Connection Manager, or as a parameter to an MQQueueManager constructor. When an MQQueueManager is constructed, the most recently used connection in the pool is used.

Connections are destroyed by a separate thread when they are unused for a specified period, when there are more than a specified number of unused connections in the pool, or when the maximum number of connections has been reached and room must be made for new connections. You can specify the timeout period, the maximum number of managed connections, and the maximum number of unused connections.

Variables

MODE_ACTIVE
public static final int MODE_ACTIVE. See “setActive” on page 177.

MODE_AUTO
public static final int MODE_AUTO. See “setActive” on page 177.

MODE_INACTIVE
public static final int MODE_INACTIVE. See “setActive” on page 177.

Constructors

MQSimpleConnectionManager
public MQSimpleConnectionManager()
Constructs an MQSimpleConnectionManager.

Methods

getActive
public int getActive()
Gets the mode of the connection pool.

Returns
The current active mode of the connection pool, with one of the following values (see “setActive” on page 177):
MODE_ACTIVE
MODE_AUTO
MODE_INACTIVE

getHighThreshold (deprecated)
public int getHighThreshold()
Do not use this method in new applications. It performs the same function as getMaxUnusedConnections and returns the maximum number of unused connections in the pool.

getMaxConnections
public int getMaxConnections()
Returns the maximum number of connections managed by the connection manager.

getMaxUnusedConnections
public int getMaxUnusedConnections()
Returns the maximum number of unused connections in the pool.

getTimeout
public long getTimeout()
Returns the timeout value.

setActive
public void setActive(int mode)
Sets the active mode of the connection pool.

Parameters
mode The required active mode of the connection pool. Valid values are:

- **MODE_ACTIVE**
  The connection pool is always active. When MQQueueManager.disconnect() is called, the underlying connection is pooled and potentially reused the next time that an MQQueueManager object is constructed. Connections are destroyed by a separate thread if they are unused for longer than the timeout period, if the number of unused connections in the pool exceeds the value set by setMaxUnusedConnections(), or if room must be made for a new connection.

- **MODE_AUTO**
  The connection pool is active while the connection manager is the default connection manager and there is at least one token in the set of MQPoolToken objects held by the MQEnvironment object. This is the default mode.

- **MODE_INACTIVE**
  The connection pool is always inactive. When this mode is entered, the pool of connections to WebSphere MQ is cleared. When MQQueueManager.disconnect() is called, the connection that underlies any active MQQueueManager object ends.

setHighThreshold (deprecated)
public void setHighThreshold(int threshold)
Do not use this method in new applications. It performs the same function as setMaxUnusedConnections and sets the maximum number of unused connections in the pool.

Parameters
threshold The maximum number of unused connections in the pool.
**setMaxConnections**

```java
public void setMaxConnections(int maxConnections)
```

Sets the maximum number of connections to be managed. To prevent this number from being exceeded, the oldest unused connection in the pool might be destroyed or a request for a new connection might be refused. If the latter event occurs, an MQException is thrown with reason code MQRC_MAX_CONNS_LIMIT_REACHED.

**Parameters**

- `maxConnections`
  - The maximum number of connections in the pool.

**setMaxUnusedConnections**

```java
public void setMaxUnusedConnections(int maxUnusedConnections)
```

Sets the maximum number of unused connections in the pool. To prevent this number from being exceeded, the oldest unused connection in the pool is destroyed.

**Parameters**

- `maxUnusedConnections`
  - The maximum number of unused connections in the pool.

**setTimeout**

```java
public void setTimeout(long timeout)
```

Sets the timeout value, where connections that remain unused for this length of time are destroyed by a separate thread.

**Parameters**

- `timeout`
  - The value of the timeout in milliseconds.
public interface MQC
extends Object

The MQC interface defines all the constants used by the WebSphere MQ Java programming interface (except for completion code constants and error code constants). To refer to one of these constants from within your programs, prefix the constant name with MQC.. For example, you can set the close options for a queue as follows:
MQQueue queue;
...
queue.closeOptions = MQC.MQCO_DELETE; // delete the
// queue when
// it is closed
...

A full description of these constants is in the WebSphere MQ Application Programming Reference

Completion code and error code constants are defined in the MQException class. See “MQException” on page 117.
MQPoolServicesEventListener

public interface MQPoolServicesEventListener extends Object

Note: Normally, applications do not use this interface.

MQPoolServicesEventListener is for implementation by providers of default ConnectionManagers. When an MQPoolServicesEventListener is registered with an MQPoolServices object, the event listener receives an event whenever an MQPoolToken is added to, or removed from, the set of MQPoolTokens that MQEnvironment manages. It also receives an event whenever the default ConnectionManager changes.

See also “MQPoolServices” on page 146 and “MQPoolServicesEvent” on page 147.

Methods

defaultConnectionManagerChanged
   public void defaultConnectionManagerChanged(MQPoolServicesEvent event)

   Called when the default ConnectionManager is set. The set of MQPoolTokens is cleared.

tokenAdded
   public void tokenAdded(MQPoolServicesEvent event)

   Called when an MQPoolToken is added to the set.

tokenRemoved
   public void tokenRemoved(MQPoolServicesEvent event)

   Called when an MQPoolToken is removed from the set.
MQConnectionManager

This is a private interface that cannot be implemented by applications. WebSphere MQ classes for Java supplies an implementation of this interface (MQSimpleConnectionManager), which you can specify on the MQQueueManager constructor, or through MQEnvironment.setDefaultConnectionManager.

See "MQSimpleConnectionManager" on page 176.

Applications or middleware that want to provide their own ConnectionManager must implement javax.resource.spi.ConnectionManager. This requires Java 2 v1.3 with JAAS 1.0 installed.
MQReceiveExit

public interface MQReceiveExit
extends Object

The receive exit interface allows you to examine and possibly alter the data received from the queue manager by the WebSphere MQ classes for Java.

Note: This interface does not apply when connecting directly to WebSphere MQ in bindings mode.

To provide your own receive exit, define a class that implements this interface. Create a new instance of your class and assign the MQEnvironment.receiveExit variable to it before constructing your MQQueueManager object. For example:

// in MyReceiveExit.java
class MyReceiveExit implements MQReceiveExit {
    // you must provide an implementation
    // of the receiveExit method
    public byte[] receiveExit(MQChannelExit channelExitParms,
                               MQChannelDefinition channelDefinition,
                               byte[] agentBuffer) {
        // your exit code goes here...
    }
}  // in your main program...
MQEnvironment.receiveExit = new MyReceiveExit();
...  // other initialization
MQQueueManager qMgr = new MQQueueManager("");

Methods

receiveExit

public abstract byte[] receiveExit(MQChannelExit channelExitParms,
                                    MQChannelDefinition channelDefinition,
                                    byte agentBuffer[])

The receive exit method that your class must provide. This method is invoked whenever the WebSphere MQ classes for Java receives some data from the queue manager.

Parameters

channelExitParms
Contains information regarding the context in which the exit is being invoked. The exitResponse member variable is an output parameter that you use to tell the WebSphere MQ classes for Java what action to take next. See "MQChannelExit" on page 104 for further details.

channelDefinition
Contains details of the channel through which all communications with the queue manager take place.

agentBuffer
If the channelExitParms.exitReason is MQChannelExit.MQXR_XMIT, agentBuffer contains the data received from the queue manager; otherwise agentBuffer is null.
Returns

If the exit response code (in channelExitParms) is set so that the WebSphere MQ classes for Java can now process the data (MQXCC_OK), your receive exit method must return the data to be processed. The simplest receive exit, therefore, consists of the single line `return agentBuffer;`.

See also:
- “MQC” on page 179
- ”MQChannelDefinition” on page 102
MQSecurityExit

public interface MQSecurityExit extends Object

The security exit interface allows you to customize the security flows that occur when an attempt is made to connect to a queue manager.

**Note:** This interface does not apply when connecting directly to WebSphere MQ in bindings mode.

To provide your own security exit, define a class that implements this interface. Create a new instance of your class and assign the MQEnvironment.securityExit variable to it before constructing your MQQueueManager object. For example:

```java
// in MySecurityExit.java
class MySecurityExit implements MQSecurityExit {
    // you must provide an implementation
    // of the securityExit method
    public byte[] securityExit(
        MQChannelExit channelExitParms,
        MQChannelDefinition channelDefinition,
        byte[] agentBuffer)
    {
        // your exit code goes here...
    }
}

// in your main program...
MQEnvironment.securityExit = new MySecurityExit();
...
// other initialization
MQQueueManager qMgr = new MQQueueManager('');
```

**Methods**

securityExit

```java
public abstract byte[] securityExit(MQChannelExit channelExitParms,
                                     MQChannelDefinition channelDefinition,
                                     byte agentBuffer[])
```

The security exit method that your class must provide.

**Parameters**

channelExitParms
Contains information regarding the context in which the exit is being invoked. The exitResponse member variable is an output parameter that you use to tell the WebSphere MQ Client for Java what action to take next. See the "MQChannelExit" on page 104 for further details.

channelDefinition
Contains details of the channel through which all communications with the queue manager take place.

agentBuffer
If the channelExitParms.exitReason is MQChannelExit.MQXR_SEC_MSG, agentBuffer contains the security message received from the queue manager; otherwise agentBuffer is null.
Returns

If the exit response code (in channelExitParms) is set so that a message is to be transmitted to the queue manager, your security exit method must return the data to be transmitted.

See also:

- “MQC” on page 179
- “MQChannelDefinition” on page 102
MQSendExit

public interface MQSendExit
extends Object

The send exit interface allows you to examine and possibly alter the data sent to
the queue manager by the WebSphere MQ Client for Java.

Note: This interface does not apply when connecting directly to WebSphere MQ in
bindings mode.

To provide your own send exit, define a class that implements this interface. Create
a new instance of your class and assign the MQEnvironment.sendExit variable to it
before constructing your MQQueueManager object. For example:

// in MySendExit.java
class MySendExit implements MQSendExit {
    // you must provide an implementation of the sendExit method
    public byte[] sendExit(MQChannelExit channelExitParms,
                            MQChannelDefinition channelDefinition,
                            byte[] agentBuffer)
    {
        // your exit code goes here...
    }
}

// in your main program...
MQEnvironment.sendExit = new MySendExit();
...  // other initialization
MQQueueManager qMgr = new MQQueueManager("");

Methods

sendExit

public abstract byte[] sendExit(MQChannelExit channelExitParms,
                                 MQChannelDefinition channelDefinition,
                                 byte agentBuffer[])

The send exit method that your class must provide. This method is
invoked whenever the WebSphere MQ classes for Java wishes to transmit
some data to the queue manager.

Parameters

channelExitParms
Contains information regarding the context in which the exit is
being invoked. The exitResponse member variable is an output
parameter that you use to tell the WebSphere MQ classes for Java
what action to take next. See “MQChannelExit” on page 104 for
further details.

channelDefinition
Contains details of the channel through which all communications
with the queue manager take place.

agentBuffer
If the channelExitParms.exitReason is
MQChannelExit.MQXR_XMIT, agentBuffer contains the data to be
transmitted to the queue manager; otherwise agentBuffer is null.
Returns

If the exit response code (in channelExitParms) is set so that a message is to be transmitted to the queue manager (MQXCC_OK), your send exit method must return the data to be transmitted. The simplest send exit, therefore, consists of the single line `return agentBuffer;`.

See also:
- “MQC” on page 179
- “MQChannelDefinition” on page 102
ManagedConnection

ManagedConnection

public interface javax.resource.spi.ManagedConnection

Note: Normally, applications do not use this class; it is intended for use by implementations of ConnectionManager.

WebSphere MQ classes for Java provides an implementation of ManagedConnection that is returned from ManagedConnectionFactory.createManagedConnection. This object represents a connection to a WebSphere MQ Queue Manager. For more details about this interface, see the J2EE Connector Architecture specification (refer to Sun’s Web site at http://java.sun.com).

Methods

addConnectionEventListener

public void addConnectionEventListener(ConnectionEventListener listener)

Adds a ConnectionEventListener to the ManagedConnection instance.

The listener is notified if a severe error occurs on the ManagedConnection, or when MQQueueManager.disconnect() is called on a connection handle that is associated with this ManagedConnection. The listener is not notified about local transaction events (see “getLocalTransaction” on page 189).

associateConnection

public void associateConnection(Object connection)

Throws ResourceException.

WebSphere MQ classes for Java does not currently support this method. A javax.resource.NotSupportedException is thrown.

cleanup

public void cleanup()

Throws ResourceException.

Closes all open connection handles, and resets the physical connection to an initial state ready to be pooled. Any pending local transaction is rolled back. For more details, see “getLocalTransaction” on page 189.

destroy

public void destroy()

Throws ResourceException.

Destroys the physical connection to the WebSphere MQ Queue Manager. Any pending local transaction is committed. For more details, see “getLocalTransaction” on page 189.

getConnection

public Object getConnection(javax.security.auth.Subject subject, ConnectionRequestInfo cxRequestInfo)

Throws ResourceException.
ManagedConnection

Creates a new connection handle for the physical connection represented by the ManagedConnection object. For WebSphere MQ classes for Java, this returns an MQQueueManager object. The ConnectionManager normally returns this object from allocateConnection.

The subject parameter is ignored. If the cxRequestInfo parameter is not suitable, a ResourceException is thrown. Multiple connection handles can be used simultaneously for each single ManagedConnection.

getLocalTransaction

public LocalTransaction getLocalTransaction()

Throws ResourceException.

WebSphere MQ classes for Java does not currently support this method. A javax.resource.NotSupportedException is thrown.

Currently, a ConnectionManager cannot manage the WebSphere MQ local transaction, and registered ConnectionEventListener are not informed about events relating to the local transaction. When cleanup() occurs, any ongoing unit of work is rolled back. When destroy() occurs, any ongoing unit of work is committed.

Existing API behavior is that an ongoing unit of work is committed at MQQueueManager.disconnect(). This existing behavior is preserved only when an MQConnectionManager (rather than a ConnectionManager) manages the connection.

getLogWriter

public java.io.PrintWriter getLogWriter()

Throws ResourceException.

Returns the log writer for this ManagedConnection.

WebSphere MQ classes for Java does not currently use the log writer. See “MQException.log” on page 117 for more information about logging.

getMetaData

public ManagedConnectionMetaData getMetaData()

Throws ResourceException.

Gets the meta data information for the underlying Queue Manager. See “ManagedConnectionMetaData” on page 193.

getXAResource

public javax.transaction.xa.XAResource getXAResource()

Throws ResourceException.

WebSphere MQ classes for Java does not currently support this method. A javax.resource.NotSupportedException is thrown.

removeConnectionEventListener

public void removeConnectionEventListener(ConnectionEventListener listener)

Removes a registered ConnectionEventListener.
ManagedConnection

setLogWriter

public void setLogWriter(java.io.PrintWriter out)

Throws ResourceException.

Sets the log writer for this ManagedConnection. When a ManagedConnection is created, it inherits the log writer from its ManagedConnectionFactory.

WebSphere MQ classes for Java does not currently use the log writer. See “MQException.log” on page 117 for more information about logging.
ManagedConnectionFactory

public interface javax.resource.spi.ManagedConnectionFactory

Note: Normally, applications do not use this class; it is intended for use by implementations of ConnectionManager.

WebSphere MQ classes for Java provides an implementation of this interface to ConnectionManagers. A ManagedConnectionFactory is used to construct ManagedConnections and to select suitable ManagedConnections from a set of candidates. For more details about this interface, see the J2EE Connector Architecture specification (refer to Sun’s Web site at http://java.sun.com).

Methods

createConnectionFactory
    public Object createConnectionFactory()

    Throws ResourceException.

    WebSphere MQ classes for Java does not currently support the createConnectionFactory methods. This method throws a javax.resource.NotSupportedException.

createConnectionFactory
    public Object createConnectionFactory(ConnectionManager cxManager)

    Throws ResourceException.

    WebSphere MQ classes for Java does not currently support the createConnectionFactory methods. This method throws a javax.resource.NotSupportedException.

createManagedConnection
    public ManagedConnection createManagedConnection
        (javax.security.auth.Subject subject,
         ConnectionRequestInfo cxRequestInfo)

    Throws ResourceException.

    Creates a new physical connection to a WebSphere MQ Queue Manager, and returns a ManagedConnection object that represents this connection. WebSphere MQ ignores the subject parameter.

equals
    public boolean equals(Object other)

    Checks whether this ManagedConnectionFactory is equal to another ManagedConnectionFactory. Returns true if both ManagedConnectionFactoryes describe the same target Queue Manager.

getLogWriter
    public java.io.PrintWriter getLogWriter()

    Throws ResourceException.

    Returns the log writer for this ManagedConnectionFactory.
WebSphere MQ classes for Java does not currently use the log writer. See “MQException.log” on page 117 for more information about logging.

hashCode

public int hashCode()

Returns the hash code for this ManagedConnectionFactory.

matchManagedConnection

public ManagedConnection matchManagedConnection
               (java.util.Set connectionSet,
                javax.security.auth.Subject subject,
                ConnectionRequestInfo cxRequestInfo)

Throws ResourceException.

Searches the supplied set of candidate ManagedConnections for an appropriate ManagedConnection. Returns either null, or a suitable ManagedConnection from the set that meets the criteria for connection.

setLogWriter

public void setLogWriter(java.io.PrintWriter out)

Throws ResourceException.

Sets the log writer for this ManagedConnectionFactory. When a ManagedConnection is created, it inherits the log writer from its ManagedConnectionFactory.

WebSphere MQ classes for Java does not currently use the log writer. See “MQException.log” on page 117 for more information about logging.
ManagedConnectionMetaData

public interface javax.resource.spi.ManagedConnectionMetaData

Note: Normally, applications do not use this interface; it is intended for use by implementations of ConnectionManager.

A ConnectionManager can use this interface to retrieve meta data that is related to an underlying physical connection to a Queue Manager. An implementation of this interface is returned from ManagedConnection.getMetaData(). For more details about this interface, see the J2EE Connector Architecture specification (refer to Sun’s Web site at http://java.sun.com).

Methods

getEISProductName

public String getEISProductName()

Throws ResourceException.

Returns IBM WebSphere MQ.

getMaxConnections

public int getMaxConnections()

Throws ResourceException.

Returns 0.

getProductVersion

public String getProductVersion()

Throws ResourceException.

Returns a string that describes the command level of the WebSphere MQ queue manager to which the ManagedConnection is connected.

getUserName

public String getUserName()

Throws ResourceException.

If the ManagedConnection represents a client connection to a queue manager, this returns the user ID used for the connection. Otherwise, it returns an empty string.
### Chapter 13. JMS messages

**Message selectors** .............................................. 257  
**Mapping JMS messages onto WebSphere MQ messages** ....... 257  
**The MQRFH2 header** ............................................ 261  
**JMS fields and properties with corresponding MQMD fields** .. 265  
**Mapping JMS fields onto WebSphere MQ fields** ............ 266  
  - Mapping JMS header fields at send() or publish() ........ 268  
  - Mapping JMS property fields ............................... 269  
  - Mapping JMS provider-specific fields ...................... 270  
**Mapping WebSphere MQ fields onto JMS fields** ............ 271  
  - (incoming messages) ...................................... 273  
  - Mapping JMS to a native WebSphere MQ application ....... 273  
  - Message body .............................................. 273

### Chapter 14. WebSphere MQ JMS Application Server Facilities

**ASF classes and functions** .................................... 277  
**ConnectionConsumer** ........................................... 277  
**Planning an application** ...................................... 278  
  - General principles for point-to-point messaging ............ 278  
  - General principles for publish/subscribe messaging ......... 279  
  - Handling poison messages ................................... 280  
  - Removing messages from the queue .......................... 281  
**Error handling** ................................................. 282  
**Recovering from error conditions** ............................ 282  
**Reason and feedback codes** .................................. 283  
**Application server sample code** .............................. 283  
**MyServerSession.java** .......................................... 285  
**MyServerSessionFactory.java** .................................. 285  
**MessageListenerFactory.java** ................................. 286  
**Examples of ASF use** .......................................... 287  
**Load1.java** ....................................................... 287  
**CountingMessageListenerFactory.java** ........................ 288  
**ASFClient1.java** ............................................... 289  
**Load2.java** ....................................................... 290  
**LoggingMessageListenerFactory.java** ........................ 290  
**ASFClient2.java** ............................................... 290  
**TopicLoad.java** .................................................. 291  
**ASFClient3.java** ............................................... 292  
**ASFClient4.java** ................................................. 293  
**ASFClient5.java** ................................................. 294

### Chapter 15. JMS interfaces and classes

**Sun Java Message Service classes and interfaces** .......... 295  
**WebSphere MQ JMS classes** ..................................... 298  
**BytesMessage** ................................................... 300  
  - Methods ...................................................... 300  
**Cleanup** .......................................................... 308  
  - WebSphere MQ constructor ................................... 308

**Methods** .......................................................... 308  
**Connection** ..................................................... 313  
**Methods** .......................................................... 313  
**ConnectionConsumer** .......................................... 318  
**Methods** .......................................................... 318  
**ConnectionFactory** ............................................. 319  
**WebSphere MQ constructor** .................................... 319  
**Methods** .......................................................... 319  
**ConnectionMetaData** .......................................... 335  
**WebSphere MQ constructor** .................................... 335  
**Methods** .......................................................... 335  
**DeliveryMode** ................................................... 337  
**Fields** ............................................................. 337  
**Destination** ...................................................... 338  
**WebSphere MQ constructors** .................................. 338  
**Methods** .......................................................... 338  
**ExceptionListener** .............................................. 340  
**Methods** .......................................................... 340  
**MapMessage** ..................................................... 341  
**Methods** .......................................................... 341  
**Message** .......................................................... 349  
**Methods** .......................................................... 349  
**MessageConsumer** .............................................. 363  
**Methods** .......................................................... 363  
**MessageListener** ................................................ 366  
**Methods** .......................................................... 366  
**MessageProducer** .............................................. 367  
**WebSphere MQ constructors** .................................. 367  
**Methods** .......................................................... 367  
**MQQueueEnumeration** ........................................... 373  
**Methods** .......................................................... 373  
**ObjectMessage** .................................................. 374  
**Methods** .......................................................... 374  
**Queue** .............................................................. 375  
**WebSphere MQ constructors** .................................. 375  
**Methods** .......................................................... 375  
**QueueBrowser** ................................................... 377  
**Methods** .......................................................... 377  
**QueueConnection** ............................................... 379  
**Methods** .......................................................... 379  
**QueueConnectionFactory** ....................................... 381  
**WebSphere MQ constructor** .................................... 381  
**Methods** .......................................................... 381  
**QueueReceiver** ................................................... 384  
**Methods** .......................................................... 384  
**QueueRequester** ................................................ 385  
**Constructors** ................................................... 385  
**Methods** .......................................................... 385  
**QueueSender** ..................................................... 387  
**Methods** .......................................................... 387  
**QueueSession** .................................................... 390  
**Methods** .......................................................... 390  
**Session** ............................................................ 393  
**Fields** ............................................................. 393  
**Methods** .......................................................... 393  
**StreamMessage** .................................................. 405  
**Methods** .......................................................... 405  
**TemporaryQueue** ............................................... 413  
**Methods** .......................................................... 413  
**TemporaryTopic** ................................................ 414
<table>
<thead>
<tr>
<th></th>
<th>Part 3. Programming with WebSphere MQ JMS</th>
<th>197</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebSphere MQ constructor.</td>
<td>414</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>414</td>
<td></td>
</tr>
<tr>
<td>TextMessage.</td>
<td>415</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>415</td>
<td></td>
</tr>
<tr>
<td>Topic</td>
<td>416</td>
<td></td>
</tr>
<tr>
<td>WebSphere MQ constructor.</td>
<td>416</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>416</td>
<td></td>
</tr>
<tr>
<td>TopicConnection</td>
<td>420</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>420</td>
<td></td>
</tr>
<tr>
<td>TopicConnectionFactory</td>
<td>423</td>
<td></td>
</tr>
<tr>
<td>WebSphere MQ constructor.</td>
<td>423</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>423</td>
<td></td>
</tr>
<tr>
<td>TopicPublisher</td>
<td>431</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>431</td>
<td></td>
</tr>
<tr>
<td>TopicRequestor</td>
<td>434</td>
<td></td>
</tr>
<tr>
<td>Constructors</td>
<td>434</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>434</td>
<td></td>
</tr>
<tr>
<td>TopicSession.</td>
<td>436</td>
<td></td>
</tr>
<tr>
<td>WebSphere MQ constructor.</td>
<td>436</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>436</td>
<td></td>
</tr>
<tr>
<td>TopicSubscriber</td>
<td>440</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>440</td>
<td></td>
</tr>
<tr>
<td>XAConnection</td>
<td>441</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>441</td>
<td></td>
</tr>
<tr>
<td>XAConnectionFactory</td>
<td>443</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>443</td>
<td></td>
</tr>
<tr>
<td>XAQueueConnection</td>
<td>445</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>445</td>
<td></td>
</tr>
<tr>
<td>XAQueueConnectionFactory</td>
<td>446</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>446</td>
<td></td>
</tr>
<tr>
<td>XASession</td>
<td>448</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>448</td>
<td></td>
</tr>
<tr>
<td>XATopicConnection</td>
<td>451</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>451</td>
<td></td>
</tr>
<tr>
<td>XATopicConnectionFactory</td>
<td>452</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>452</td>
<td></td>
</tr>
<tr>
<td>XATopicSession</td>
<td>454</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>454</td>
<td></td>
</tr>
</tbody>
</table>
This chapter provides information to help with writing WebSphere MQ JMS applications. It gives a brief introduction to the JMS model, and detailed information on programming some common tasks that application programs are likely to need to perform.

The JMS model

JMS defines a generic view of a message passing service. The generic JMS model is based around the following interfaces that are defined in Sun’s java.jms package:

**Connection**
Provides access to the underlying transport, and is used to create Sessions.

**Session**
Provides a context for producing and consuming messages, including the methods used to create MessageProducers and MessageConsumers.

**MessageProducer**
Used to send messages.

**MessageConsumer**
Used to receive messages.

A Connection is thread safe, but Sessions, MessageProducers, and MessageConsumers are not. The recommended strategy is to use one Session per application thread.

In WebSphere MQ terms:

**Connection**
Provides a scope for temporary queues. Also, it provides a place to hold the parameters that control how to connect to WebSphere MQ. Examples of these parameters are the name of the queue manager, and the name of the remote host if you use the WebSphere MQ Java client connectivity.

**Session**
Contains an HCONN and therefore defines a transactional scope.

**MessageProducer and MessageConsumer**
Contain an HOBJ that defines a particular queue for writing to or reading from.

Note that normal WebSphere MQ rules apply:

- Only a single operation can be in progress per HCONN at any given time. Therefore, the MessageProducers or MessageConsumers associated with a Session cannot be called concurrently. This is consistent with the JMS restriction of a single thread per Session.
- PUTs can use remote queues, but GETs can only be applied to queues on the local queue manager.

The generic JMS interfaces are subclassed into more specific versions for point-to-point and publish/subscribe behavior.

The point-to-point versions are:
JMS model

- QueueConnection
- QueueSession
- QueueSender
- QueueReceiver

When using JMS, always write application programs that use only references to the interfaces in `javax.jms`. All vendor-specific information is encapsulated in implementations of:
- QueueConnectionFactory
- TopicConnectionFactory
- Queue
- Topic

These are known as *administered objects*, that is, objects that can be built using a vendor-supplied administration tool and stored in a JNDI namespace. A JMS application can retrieve these objects from the namespace and use them without needing to know which vendor provided the implementation.

Building a connection

Connections are not created directly, but are built using a connection factory. Factory objects can be stored in a JNDI namespace, insulating the JMS application from provider-specific information. Details of how to create and store factory objects are in Chapter 5, “Using the WebSphere MQ JMS administration tool,” on page 41.

If you do not have a JNDI namespace available, see “Creating factories at runtime” on page 201.

Retrieving the factory from JNDI

To retrieve an object from a JNDI namespace, set up an initial context, as shown in this fragment taken from the IVTRun sample file:

```java
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;

java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
environment.put(Context.PROVIDER_URL, url);
Context ctx = new InitialDirContext(environment);
```

where:

- `icf` defines a factory class for the initial context
- `url` defines a context specific URL

For more details about JNDI usage, see Sun’s JNDI documentation.

Note: Some combinations of the JNDI packages and LDAP service providers can result in an LDAP error 84. To resolve the problem, insert the following line before the call to `InitialDirContext`.

```java
environment.put(Context.REFERRAL, "throw");
```
Once an initial context is obtained, objects are retrieved from the namespace by using the lookup() method. The following code retrieves a QueueConnectionFactory named ivtQCF from an LDAP-based namespace:

```java
QueueConnectionFactory factory;
factory = (QueueConnectionFactory)ctx.lookup("cn=ivtQCF");
```

**Using the factory to create a connection**

The createQueueConnection() method on the factory object is used to create a Connection, as shown in the following code:

```java
QueueConnection connection;
connection = factory.createQueueConnection();
```

**Creating factories at runtime**

If a JNDI namespace is not available, it is possible to create factory objects at runtime. However, using this method reduces the portability of the JMS application because it requires references to WebSphere MQ specific classes.

The following code creates a QueueConnectionFactory with all default settings:

```java
factory = new com.ibm.mq.jms.MQQueueConnectionFactory();
```

(You can omit the com.ibm.mq.jms. prefix if you import the com.ibm.mq.jms package instead.)

A connection created from the above factory uses the Java bindings to connect to the default queue manager on the local machine. The set methods shown in Table 14 on page 202 can be used to customize the factory with WebSphere MQ specific information.

The only way to create a TopicConnectionFactory object at runtime is to construct it using the MQTopicConnectionFactory constructor. For example:

```java
MQTopicConnectionFactory fact = new MQTopicConnectionFactory();
```

This creates a default TopicConnectionFactory object with the bindings transportType and all other default settings.

It is possible to change the transportType for the TopicConnectionFactory using its setTransportType() method. For example:

```java
fact.setTransportType(JMSC.MQJMS_TP_BINDINGS_MQ); // Bindings mode
fact.setTransportType(JMSC.MQJMS_TP_CLIENT_MQ_TCP); // Client mode
fact.setTransportType(JMSC.MQJMS_TP_DIRECT_TCP); // Direct TCP/IP mode
```

The full JMS TopicConnectionFactory interface has been implemented. Refer to “TopicConnectionFactory” on page 423 for more details. Note that certain combinations of property settings are not valid for TopicConnectionFactory objects. See “Properties” on page 49 for more details.

**Starting the connection**

The JMS specification defines that connections should be created in the stopped state. Until the connection starts, MessageConsumers that are associated with the connection cannot receive any messages. To start the connection, issue the following command:

```java
connection.start();
```
Choosing client or bindings transport

WebSphere MQ JMS can communicate with WebSphere MQ using either the client or bindings transports. (However, client transport is not supported on the z/OS and OS/390 platforms.) If you use the Java bindings, the JMS application and the WebSphere MQ queue manager must be located on the same machine. If you use the client, the queue manager can be on a different machine from the application.

The contents of the connection factory object determine which transport to use. Chapter 5, “Using the WebSphere MQ JMS administration tool,” on page 41 describes how to define a factory object for use with client or bindings transport.

The following code fragment illustrates how you can define the transport within an application:

```java
String HOSTNAME = "machine1";
String QMGRNAME = "machine1.QM1";
String CHANNEL = "SYSTEM.DEF.SVRCONN";

MQQueueConnectionFactory factory = new MQQueueConnectionFactory();
```

### Table 14. Set methods on MQQueueConnectionFactory

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setCCSID(int)</td>
<td>Used to set the MQEnvironment.CCSID property</td>
</tr>
<tr>
<td>setChannel(String)</td>
<td>The name of the channel for a client connection</td>
</tr>
</tbody>
</table>
| setFailIfQuiesce(int)         | Defines the behavior an application exhibits when making calls (for example, send and receive) against a quiescing queue manager. The options are:  
  • JMSC.MQJMS_FIQ_NO
  • JMSC.MQJMS_FIQ_YES (the default)                                                             |
| setHostName(String)           | The name of the host for a client connection                                                    |
| setPort(int)                  | The port for a client connection                                                                |
| setQueueManager(String)       | The name of the queue manager                                                                  |
| setTemporaryModel(String)     | The name of a model queue used to generate a temporary destination as a result of a call to QueueSession.createTemporaryQueue(). Make this the name of a temporary dynamic queue, rather than a permanent dynamic queue. |
| setTempQPrefix(String)        | The prefix that is used to form the name of a WebSphere MQ dynamic queue.                     |
| setTransportType(int)         | How to connect to WebSphere MQ. The options are:                                                |
|                               | • JMSC.MQJMS_TP_BINDINGS_MQ (the default)                                                       |
|                               | • JMSC.MQJMS_TP_CLIENT_MQ_TCP/IP                                                               |
|                               | JMSC is in the package com.ibm.mq.jms                                                          |
| setReceiveExit(String)        | Allow the use of the send, receive, and security exits provided by the underlying WebSphere MQ Classes for Java. The setExit methods take the name of a class that implements the relevant exit methods. (See the WebSphere MQ product documentation for details.) |
| setSecurityExit(String)       |                                                                                               |
| setSendExit(String)           |                                                                                               |
| setReceiveExitInit(String)    |                                                                                               |
| setSecurityExitInit(String)   |                                                                                               |
| setSendExitInit(String)       |                                                                                               |
factory.setTransportType(JMSC.MQJMS_TP_CLIENT_MQ_TCPPIP);
factory.setQueueManager(QMGRNAME);
factory.setHostName(HOSTNAME);
factory.setChannel(CHANNEL);

### Specifying a range of ports for client connections

If a JMS application attempts to connect to a WebSphere MQ queue manager in client mode, a firewall might allow only those connections that originate from specified ports or a range of ports. In this situation, you can use the LOCALADDRESS property of a QueueConnectionFactory or TopicConnectionFactory object to specify a port, or a range of points, that the application can bind to.

You can set the LOCALADDRESS property by using the WebSphere MQ JMS administration tool, or by calling the setLocalAddress() method in a JMS application. Here is an example of setting the property from within an application:

```java
mqConnectionFactory.setLocalAddress("9.20.0.1(2000,3000)");
```

When the application connects to a queue manager subsequently, the application binds to a local IP address and port number in the range 9.20.0.1(2000) to 9.20.0.1(3000).

Connection errors might occur if you restrict the range of ports. If an error occurs, a JMSExcetion is thrown with an embedded MQException that contains the WebSphere MQ reason code, MQRC_Q_MGR_NOT_AVAILABLE. An error might occur if all the ports in the specified range are in use, or if the LOCALADDRESS property contains an IP address, host name, or port number that is not valid; a negative port number, for example.

Because the WebSphere MQ JMS client might create connections other than those required by an application, always consider specifying a range of ports. In general, every Session created by an application requires one port and the WebSphere MQ JMS client might require three additional ports. If a connection error does occur, increase the range of ports.

JMS connection pooling might have an effect on the speed at which ports can be reused. As a result, a connection error might occur while ports are being freed.

### Obtaining a session

Once a connection is made, use the createQueueSession method on the QueueConnection to obtain a session.

The method takes two parameters:
1. A boolean that determines whether the session is transacted or non-transacted.
2. A parameter that determines the acknowledge mode.
Sending a message

Messages are sent using a MessageProducer. For point-to-point this is a QueueSender that is created using the createSender method on QueueSession. A QueueSender is normally created for a specific queue, so that all messages sent using that sender are sent to the same destination. The destination is specified using a Queue object. Queue objects can be either created at runtime, or built and stored in a JNDI namespace.

Queue objects are retrieved from JNDI in the following way:

```java
Queue ioQueue;
ioQueue = (Queue)ctx.lookup( qLookup );
```

WebSphere MQ JMS provides an implementation of Queue in com.ibm.mq.jms.MQQueue. It contains properties that control the details of WebSphere MQ specific behavior, but in many cases it is possible to use the default values. JMS defines a standard way to specify the destination that minimizes the WebSphere MQ specific code in the application. This mechanism uses the QueueSession.createQueue method, which takes a string parameter describing the destination. The string itself is still in a vendor-specific format, but this is a more flexible approach than directly referring to the vendor classes.

WebSphere MQ JMS accepts two forms for the string parameter of createQueue().

- The first is the name of the WebSphere MQ queue, as illustrated in the following fragment taken from the IVTRun program in the samples directory:
  ```java
  public static final String QUEUE = "SYSTEM.DEFAULT.LOCAL.QUEUE" ;
  .
  .
  .
  ioQueue = session.createQueue( QUEUE );
  ```

- The second, and more powerful, form is based on uniform resource identifiers (URIs). This form allows you to specify remote queues (queues on a queue manager other than the one to which you are connected). It also allows you to set the other properties contained in a com.ibm.mq.jms.MQQueue object.

The URI for a queue begins with the sequence queue://, followed by the name of the queue manager on which the queue resides. This is followed by a further /, the name of the queue, and optionally, a list of name-value pairs that set the remaining Queue properties. For example, the URI equivalent of the previous example is:

```java
ioQueue = session.createQueue("queue:///SYSTEM.DEFAULT.LOCAL.QUEUE");
```

The name of the queue manager is omitted. This is interpreted as the queue manager to which the owning QueueConnection is connected at the time when the Queue object is used.
When sending a message to a cluster, leave the Queue Manager field in the JMS Queue object blank. This enables an MQOPEN to be performed in BIND_NOT_FIXED mode, which allows the queue manager to be determined. Otherwise an exception is returned reporting that the queue object cannot be found. This applies when using JNDI or defining queues at runtime.

The following example connects to queue Q1 on queue manager HOST1.QM1, and causes all messages to be sent as non-persistent and priority 5:

```java
ioQueue = session.createQueue("queue://HOST1.QM1/Q1?persistence=1&priority=5");
```

The following is an example of creating a topic URI:

```java
session.createTopic("topic://Sport/Football/Results?multicast=7");
```

Table 15 lists the names that can be used in the name-value part of the URI. A disadvantage of this format is that it does not support symbolic names for the values, so where appropriate, the table also indicates special values, which might change. (See “Setting properties with the set method” on page 206 for an alternative way of setting properties.)

**Table 15. Property names for queue and topic URIs**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCSID</td>
<td>Character set of the destination</td>
<td>integers - valid values listed in base WebSphere MQ documentation</td>
</tr>
<tr>
<td>encoding</td>
<td>How to represent numeric fields</td>
<td>An integer value as described in the base WebSphere MQ documentation</td>
</tr>
<tr>
<td>expiry</td>
<td>Lifetime of the message in milliseconds</td>
<td>0 for unlimited, positive integers for timeout (ms)</td>
</tr>
<tr>
<td>multicast</td>
<td>Sets multicast mode for direct connections</td>
<td>-1=ASCF, 0=DISABLED, 3=NOTR, 5=RELIABLE, 7=ENABLED</td>
</tr>
<tr>
<td>persistence</td>
<td>Whether the message should be hardened to disk</td>
<td>1=non-persistent, 2=persistent, -1=QDEF, -2=APP</td>
</tr>
<tr>
<td>priority</td>
<td>Priority of the message</td>
<td>0 through 9, -1=QDEF, -2=APP</td>
</tr>
<tr>
<td>targetClient</td>
<td>Whether the receiving application is JMS compliant</td>
<td>0=JMS, 1=MQ</td>
</tr>
</tbody>
</table>

The special values are:

- **QDEF** Determine the property from the configuration of the WebSphere MQ queue.
- **APP** The JMS application can control this property.

Once the Queue object is obtained (either using createQueue as above or from JNDI), it must be passed into the createSender method to create a QueueSender:

```java
QueueSender queueSender = session.createSender(ioQueue);
```

The resulting queueSender object is used to send messages by using the send method:

```java
queueSender.send(outMessage);
```
Setting properties with the set method

You can set Queue properties by first creating an instance of `com.ibm.mq.jms.MQQueue` using the default constructor. Then you can fill in the required values by using public set methods. This method means that you can use symbolic names for the property values. However, because these values are vendor-specific, and are embedded in the code, the applications become less portable.

The following code fragment shows the setting of a queue property with a set method.

```java
com.ibm.mq.jms.MQQueue q1 = new com.ibm.mq.jms.MQQueue();
q1.setBaseQueueManagerName("HOST1.QM1");
q1.setBaseQueueName("Q1");
q1.setPersistence(DeliveryMode.NON_PERSISTENT);
q1.setPriority(5);
```

Table 16 shows the symbolic property values that are supplied with WebSphere MQ JMS for use with the set methods.

<table>
<thead>
<tr>
<th>Property</th>
<th>Admin tool keyword</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>expiry</td>
<td>UNLIM</td>
<td>JMSC.MQJMS_EXP_UNLIMITED</td>
</tr>
<tr>
<td></td>
<td>APP</td>
<td>JMSC.MQJMS_EXP_APP</td>
</tr>
<tr>
<td>priority</td>
<td>APP</td>
<td>JMSC.MQJMS_PRI_APP</td>
</tr>
<tr>
<td></td>
<td>QDEF</td>
<td>JMSC.MQJMS_PRI_QDEF</td>
</tr>
<tr>
<td>persistence</td>
<td>APP</td>
<td>JMSC.MQJMS_PER_APP</td>
</tr>
<tr>
<td></td>
<td>QDEF</td>
<td>JMSC.MQJMS_PER_QDEF</td>
</tr>
<tr>
<td></td>
<td>PERS</td>
<td>JMSC.MQJMS_PER_PER</td>
</tr>
<tr>
<td></td>
<td>NON</td>
<td>JMSC.MQJMS_PER_NON</td>
</tr>
<tr>
<td>targetClient</td>
<td>JMS</td>
<td>JMSC.MQJMS_CLIENT_JMS_COMPLIANT</td>
</tr>
<tr>
<td></td>
<td>MQ</td>
<td>JMSC.MQJMS_CLIENT_NONJMS_MQ</td>
</tr>
<tr>
<td>encoding</td>
<td>Integer(N)</td>
<td>JMSC.MQJMS_ENCODING_INTEGER_NORMAL</td>
</tr>
<tr>
<td></td>
<td>Integer(R)</td>
<td>JMSC.MQJMS_ENCODING_INTEGER_REVERSED</td>
</tr>
<tr>
<td></td>
<td>Decimal(N)</td>
<td>JMSC.MQJMS_ENCODING_DECIMAL_NORMAL</td>
</tr>
<tr>
<td></td>
<td>Decimal(R)</td>
<td>JMSC.MQJMS_ENCODING_DECIMAL_REVERSED</td>
</tr>
<tr>
<td></td>
<td>Float(N)</td>
<td>JMSC.MQJMS_ENCODING_FLOAT_IEEE_NORMAL</td>
</tr>
<tr>
<td></td>
<td>Float(R)</td>
<td>JMSC.MQJMS_ENCODING_FLOAT_IEEE_REVERSED</td>
</tr>
<tr>
<td></td>
<td>Native</td>
<td>JMSC.MQJMS_ENCODING_NATIVE</td>
</tr>
<tr>
<td>multicast</td>
<td>ASCF</td>
<td>JMSC.MQJMS_MULTICAST_AS_CF</td>
</tr>
<tr>
<td></td>
<td>DISABLED</td>
<td>JMSC.MQJMS_MULTICAST_DISABLED</td>
</tr>
<tr>
<td></td>
<td>NOTR</td>
<td>JMSC.MQJMS_MULTICAST_NOT_RELIABLE</td>
</tr>
<tr>
<td></td>
<td>RELIABLE</td>
<td>JMSC.MQJMS_MULTICAST_RELIABLE</td>
</tr>
<tr>
<td></td>
<td>ENABLED</td>
<td>JMSC.MQJMS_MULTICAST_ENABLED</td>
</tr>
</tbody>
</table>

See “The ENCODING property” on page 57 for a discussion of encoding.

Message types

JMS provides several message types, each of which embodies some knowledge of its content. To avoid referring to the vendor-specific class names for the message types, methods are provided on the Session object for message creation.

In the sample program, a text message is created in the following manner:
System.out.println("Creating a TextMessage");
TextMessage outMessage = session.createTextMessage();
System.out.println("Adding Text");
outMessage.setText(outString);

The message types that can be used are:
- BytesMessage
- MapMessage
- ObjectMessage
- StreamMessage
- TextMessage

Details of these types are in Chapter 15, “JMS interfaces and classes,” on page 295.

Receiving a message

Messages are received using a QueueReceiver. This is created from a Session by using the createReceiver() method. This method takes a Queue parameter that defines from where the messages are received. See “Sending a message” on page 204 for details of how to create a Queue object.

The sample program creates a receiver and reads back the test message with the following code:

QueueReceiver queueReceiver = session.createReceiver(ioQueue);
Message inMessage = queueReceiver.receive(1000);

The parameter in the receive call is a timeout in milliseconds. This parameter defines how long the method should wait if there is no message available immediately. You can omit this parameter, in which case, the call blocks indefinitely. If you do not want any delay, use the receiveNoWait() method.

The receive methods return a message of the appropriate type. For example, if a TextMessage is put on a queue, when the message is received the object that is returned is an instance of TextMessage.

To extract the content from the body of the message, it is necessary to cast from the generic Message class (which is the declared return type of the receive methods) to the more specific subclass, such as TextMessage. If the received message type is not known, you can use the instanceof operator to determine which type it is. It is good practice always to test the message class before casting, so that unexpected errors can be handled gracefully.

The following code illustrates the use of instanceof, and extraction of the content from a TextMessage:

if (inMessage instanceof TextMessage) {
    String replyString = ((TextMessage) inMessage).getText();
    
} else {
    // Print error message if Message was not a TextMessage.
    System.out.println("Reply message was not a TextMessage");
}

Message selectors

JMS provides a mechanism to select a subset of the messages on a queue so that this subset is returned by a receive call. When creating a QueueReceiver, you can
Receiving a message

provide a string that contains an SQL (Structured Query Language) expression to
determine which messages to retrieve. The selector can refer to fields in the JMS
message header as well as fields in the message properties (these are effectively
application-defined header fields). Details of the header field names, as well as the
syntax for the SQL selector, are in Chapter 13, "JMS messages," on page 257.

The following example shows how to select for a user-defined property named
myProp:

queueReceiver = session.createReceiver(ioQueue, "myProp = 'blue'");

Note: The JMS specification does not permit the selector associated with a receiver
to be changed. Once a receiver is created, the selector is fixed for the lifetime
of that receiver. This means that, if you require different selectors, you must
create new receivers.

Asynchronous delivery

An alternative to making calls to QueueReceiver.receive() is to register a method
that is called automatically when a suitable message is available. The following
fragment illustrates the mechanism:

import javax.jms.*;

public class MyClass implements MessageListener
{
    // The method that will be called by JMS when a message
    // is available.
    public void onMessage(Message message)
    {
        System.out.println("message is "+message);
        // application specific processing here
        .
        .
        .
    }
}
.
.
.
// In Main program (possibly of some other class)
MyClass listener = new MyClass();
queueReceiver.setMessageListener(listener);
// main program can now continue with other application specific
// behavior.

Note: Use of asynchronous delivery with a QueueReceiver marks the entire
Session as asynchronous. It is an error to make an explicit call to the receive
methods of a QueueReceiver that is associated with a Session that is using
asynchronous delivery.

Closing down

Garbage collection alone cannot release all WebSphere MQ resources in a timely
manner, especially if the application needs to create many short-lived JMS objects
at the Session level or lower. It is therefore important to call the close() methods
of the various classes (QueueConnection, QueueSession, QueueSender, and
QueueReceiver) when the resources are no longer required.
Java Virtual Machine hangs at shutdown

If an application using WebSphere MQ JMS finishes without calling Connection.close(), some JVMs appear to hang. If this problem occurs, either edit the application to include a call to Connection.close(), or terminate the JVM using the Ctrl-C keys.

Handling errors

Any runtime errors in a JMS application are reported by exceptions. The majority of methods in JMS throw JMSExceptions to indicate errors. It is good programming practice to catch these exceptions and display them on a suitable output.

A JMSException can contain a further exception embedded in it. For JMS, this can be a valuable way to pass important detail from the underlying transport. In the case of WebSphere MQ JMS, when WebSphere MQ raises an MQException, this exception is usually included as the embedded exception in a JMSException.

The implementation of JMSException does not include the embedded exception in the output of its toString() method. Therefore, it is necessary to check explicitly for an embedded exception and print it out, as shown in the following fragment:

```
try {
    // code which may throw a JMSException

} catch (JMSException je) {
    System.err.println("caught "+je);
    Exception e = je.getLinkedException();
    if (e != null) {
        System.err.println("linked exception: "+e);
    }
}
```

Exception listener

For asynchronous message delivery, the application code cannot catch exceptions raised by failures to receive messages. This is because the application code does not make explicit calls to receive() methods. To cope with this situation, it is possible to register an ExceptionListener, which is an instance of a class that implements the onException() method. When a serious error occurs, this method is called with the JMSException passed as its only parameter. Further details are in Sun’s JMS documentation.

User exits

WebSphere MQ JMS allows you to code and use implementations of the WebSphere MQ base Java send, receive, and security exits. For WebSphere MQ JMS, ensure that your exit has a constructor that takes a single string argument. See the description of exit-related set methods in Table 14 on page 202 and "Property dependencies" on page 56.
Using Secure Sockets Layer (SSL)

Using SSL

WebSphere MQ base Java client applications and WebSphere MQ JMS connections using TRANSPORT(CLIENT) support Secure Sockets Layer (SSL) encryption. SSL provides communication encryption, authentication, and message integrity. It is typically used to secure communications between any two peers on the Internet or within an intranet.

WebSphere MQ classes for Java uses Java Secure Socket Extension (JSSE) to handle SSL encryption, and so requires a JSSE provider. J2SE v1.4 JVMs have a JSSE provider built in. Details of how to manage and store certificates can vary from provider to provider. For information about this, refer to your JSSE provider’s documentation.

This section assumes that your JSSE provider is correctly installed and configured, and that suitable certificates have been installed and made available to your JSSE provider.

SSL administrative properties

This section introduces the SSL administrative properties, as follows:

- “SSL_CIPHERSUITES object property”
- “SSL_PEERNAME object property”
- “SSL_KEYSTORES object property” on page 211
- “SSL_SOCKET_FACTORY object property” on page 212

SSL_CIPHERSUITES object property

To enable SSL encryption on a ConnectionFactory, use JMSAdmin to set the SSL_CIPHERSUITES property to a CipherSuite supported by your JSSE provider. This must match the CipherSpec set on the target channel. However, CipherSuites are distinct from CipherSpecs and so have different names. Appendix H, “SSL CipherSuites supported by WebSphere MQ,” on page 487 contains a table mapping the CipherSpecs supported by WebSphere MQ to their equivalent CipherSuites as known to JSSE. Additionally, the named CipherSuite must be supported by your JSSE provider. For more information about CipherSpecs and CipherSuites with WebSphere MQ, see the WebSphere MQ Security book.

For example, to set a QueueConnectionFactory to connect to an SSL-enabled SVRCONN channel using a CipherSpec of RC4_MD5_EXPORT, issue the following command to JMSAdmin:

`ALTER QCF(my.qcf) SSL_CIPHERSUITES(SSL_RSA_EXPORT_WITH_RC4_MD5)`

This can also be set from a program, using the setSSLCipherSuite() method on MQConnectionFactory.

For convenience, if a CipherSpec is specified on the SSL_CIPHERSUITES property, JMSAdmin attempts to map the CipherSpec to an appropriate CipherSuite and issues a warning. This attempt to map is not made if the property is specified by a program.

SSL_PEERNAME object property

A JMS application can ensure that it has connected to the correct queue manager, by specifying a distinguished name (DN) pattern. The connection succeeds only if
the queue manager presents a DN that matches the pattern. For more details of the
format of this pattern, refer to [WebSphere MQ Security] or the [WebSphere MQ Script
(MQSC) Command Reference]

The DN is set using the SSLPEERNAME property of ConnectionFactory. For
example, the following JMSAdmin command sets the ConnectionFactory to expect
the queue manager to identify itself with a Common Name beginning QMGR, with
at least two Organizational Unit names, the first of which must be IBM and the
second WEBSPHERE:

ALTER QCF(my.qcf) SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)

Checking is case-insensitive, and semicolons can be used in place of the commas.
This can also be set from a program, using the setSSLPeerName() method on
MQConnectionFactory. If this property is not set, no checking is performed on the
Distinguished Name supplied by the queue manager. This property is ignored if
no CipherSuite is set.

SSLCERTSTORES object property
It is common to use a certificate revocation list (CRL) to manage revocation of
certificates that have become untrusted. These are typically hosted on LDAP
servers; JMS allows an LDAP server to be specified for CRL checking under Java 2
v1.4 or later. The following JMSAdmin example directs JMS to use a CRL hosted
on an LDAP server named crl1.ibm.com:

ALTER QCF(my.qcf) SSLCRL(ldap://crl1.ibm.com)

Note: To use a CertStore successfully with a CRL hosted on an LDAP server, make
sure that your Java Software Development Kit (SDK) is compatible with the
CRL. Some SDKs require that the CRL conforms to RFC 2587, which defines
a schema for LDAP v2. Most LDAP v3 servers use RFC 2256 instead.

If your LDAP server is not running on the default port of 389, the port can be
specified by appending a colon and the port number to the host name. If the
certificate presented by the queue manager is present in the CRL hosted on
crl1.ibm.com, the connection does not complete. To avoid single-point-of-failure,
JMS allows multiple LDAP servers to be supplied, by supplying a space-delimited
list of LDAP servers. For example:

ALTER QCF(my.qcf) SSLCRL(ldap://crl1.ibm.com ldap://crl2.ibm.com)

When multiple LDAP servers are specified, JMS tries each one in turn until it finds
a server with which it can successfully verify the queue manager’s certificate. Each
server must contain identical information.

A string of this format can be supplied by a program on the
MQConnectionFactory.setSSLCertStores() method. Alternatively, the application can
create one or more java.security.cert.CertStore objects, place these in a suitable
Collection object, and supply this Collection to the setSSLCertStores() method. In
this way, the application can customize CRL checking. Refer to your JSSE
documentation for details on constructing and using CertStore objects.

The certificate presented by the queue manager when a connection is being set up
is validated as follows:
1. The first CertStore object in the Collection identified by sslCertStores is used to
   identify a CRL server.
2. An attempt is made to contact the CRL server.
3. If the attempt is successful, the server is searched for a match for the certificate.
Using SSL

a. If the certificate is found to be revoked, the search process is over and the
connection request fails with reason code
MQRC_SSL_CERTIFICATE_REVOKED.
b. If the certificate is not found, the search process is over and the connection
is allowed to proceed.

4. If the attempt to contact the server is unsuccessful, the next CertStore object is
used to identify a CRL server and the process repeats from step 2.

If this was the last CertStore in the Collection, or if the Collection contains no
CertStore objects, the search process has failed and the connection request fails
with reason code MQRC_SSL_CERT_STORE_ERROR.

The Collection object determines the order in which CertStores are used.

If your application uses setSSLCertStores() to set a Collection of CertStore objects,
the MQConnectionFactory can no longer be bound into a JNDI namespace.
Attempting to do so causes an exception. If the sslCertStores property is not set, no
revocation checking is performed on the certificate provided by the queue
manager. This property is ignored if no CipherSuite is set.

**SSLSocketFactory object property**

You might want to customize other aspects of the SSL connection for an
application. For example, you might want to initialize cryptographic hardware or
change the KeyStore and TrustStore in use. To do this, the application must first
create a javax.net.ssl.SSLSocketFactory instance customized accordingly. Refer to
your JSSE documentation for information on how to do this, as the customizable
features vary from provider to provider. Once a suitable SSLSocketFactory has
been obtained, use the MQConnectionFactory.setSSLSocketFactory() method to
configure JMS to use the customized SSLSocketFactory.

If your application uses setSSLSocketFactory() to set a customized
SSLSocketFactory, the MQConnectionFactory can no longer be bound into a JNDI
namespace. Attempting to do so causes an exception. If this property is not set, the
default SSLSocketFactory is used; refer to your JSSE documentation for details on
the behavior of the default SSLSocketFactory. This property is ignored if no
CipherSuite is set.

**Important:** Do not assume that use of the SSL properties ensures security when
the ConnectionFactory is retrieved from a JNDI namespace that is not
itself secure. Specifically, the standard LDAP implementation of JNDI is
not secure; an attacker can imitate the LDAP server, misleading a JMS
application into connecting to the wrong server without noticing. With
suitable security arrangements in place, other implementations of JNDI
(such as the fsexcontext implementation) are secure.
Chapter 11. Writing WebSphere MQ JMS publish/subscribe applications

You can write applications with WebSphere MQ JMS using two programming models:
- Point-to-point
- Publish/subscribe

This section considers publish/subscribe and how publish/subscribe messaging is implemented in WebSphere MQ JMS.

Introduction

With publish/subscribe messaging, one message producer can send messages to many message consumers at one time. The message producer need know nothing about the consumers receiving its messages, it needs to know only about the common destination. Similarly, the message consumers need to know only about the common destination. This common destination is called a topic.

A message producer that sends messages to a topic is a publisher and a message consumer that receives messages from a topic is a subscriber.

A message consumer receives messages on all topics to which it has subscribed. To receive messages from a topic, a message consumer must first subscribe to that topic. All messages sent to a topic are forwarded to all the message consumers subscribed to that topic at that time. Each consumer receives its own copy of each message.

JMS clients can establish durable subscriptions that allow consumers to disconnect and later reconnect and collect messages published while they were disconnected.

The connection between messages issued by publishers and the subscribers is made, in WebSphere MQ, by the publish/subscribe broker. The broker (sometimes referred to as the message broker) has a record of all the subscribers registered to a topic. When a message is published to a topic, the broker manages the forwarding of that message to the topic’s subscribers.

To run a WebSphere MQ JMS publish/subscribe application, you must be able to connect to a message broker.

Getting started with WebSphere MQ JMS and publish/subscribe

Before you can start developing publish/subscribe applications, you need to choose the broker to use and set that broker up to run the WebSphere MQ JMS.

Choosing a broker

WebSphere MQ offers a choice of three brokers:
- The MQSeries Publish/Subscribe broker uses WebSphere MQ and a SupportPac. The SupportPac MA0C is available for download from: http://www.ibm.com/software/ts/mqseries/txppacs
Getting started with publish/subscribe

If you want to use the broker-based subscription store, you must use WebSphere MQ with the MQSeries Publish/Subscribe broker. No other combination of queue manager and broker supports this store.

- WebSphere MQ Integrator provides a broker that can be run in one of two modes. Compatibility mode, which provides a broker of equivalent functionality to the MQSeries Publish/Subscribe broker; and native mode, which provides additional functionality. WebSphere MQ JMS can connect to WebSphere MQ Integrator in native mode with JMS Version 5.2.1 and later. With earlier JMS versions, it can connect to WebSphere MQ Integrator in compatibility mode only.
- WebSphere MQ Event Broker Version 2.1, WebSphere Business Integration Event Broker Version 5.0, and WebSphere Business Integration Message Broker Version 5.0 each provide a broker that can be connected to in two different ways:

Using message queues and WebSphere MQ

With this connection, you can run the broker in either compatibility mode or native mode.

Directly using a TCP/IP socket

With this connection, you can run the broker only in native mode. Also there is no support for:
- Persistent messages
- Transacted messages
- Durable subscriptions

This has implications for the implementation of the JMS specification for direct connections to the WebSphere MQ Event Broker:
- Because there are no persistent messages, JMSDeliveryMode is always NON_PERSISTENT and JMSExpiration has no meaning on messages received on direct connections.
- Because there are no transacted messages, JMSRedelivered has no meaning on messages received on direct connections.

Refer to Chapter 15, “JMS interfaces and classes,” on page 295 for specific information on each publish and subscribe interface.

Setting up the broker to run the WebSphere MQ JMS

Broker setup depends on the broker you intend to use and how you intend to use it. Each broker provides its own documentation describing installation and setup. However, for convenience and because of WebSphere MQ JMS requirements, some setup instructions are given here.

Connecting to your broker using WebSphere MQ

This section applies to the MQSeries Publish/Subscribe broker and the broker in WebSphere MQ Integrator. It also applies to the WebSphere MQ Event Broker when you choose to connect to it using WebSphere MQ.

Each broker requires its own queue manager. Refer to the broker’s documentation regarding installation and setup.

For the WebSphere MQ JMS publish/subscribe implementation to work correctly, a number of system queues must be created on the queue manager on which the broker is running. Create these message queues on each queue manager for each broker you want to run WebSphere MQ JMS. WebSphere MQ JMS provides a script that creates these queues (see “Create the WebSphere MQ JMS system queues” on page 27).
Getting started with publish/subscribe

Run the script to create the system queues. If you are using the MQSeries Publish/Subscribe broker, your broker is now fully configured. To check that the broker is correctly configured, run the publish/subscribe verification as described in “Publish/subscribe verification without JNDI” on page 35.

If you are using the broker provided by WebSphere MQ Integrator or WebSphere MQ Event Broker, configure a publish/subscribe message flow in the broker for messages to be correctly routed. The method for creating the required message flow is similar in both cases. Refer to Appendix D, “Connecting to other products,” on page 469 for details.

Connecting to your broker directly
This is possible only when you use the broker provided in WebSphere MQ Event Broker, WebSphere Business Integration Event Broker, or WebSphere Business Integration Message Broker. Because the connection to this broker is made directly, no system queues are required. However, you must set up a publish/subscribe message flow in the broker for messages to be correctly routed. Refer to Appendix D, “Connecting to other products,” on page 469 for details.

Writing a simple publish/subscribe application connecting through WebSphere MQ

This section provides a walkthrough of a simple WebSphere MQ JMS application.

Here is the complete example. Individual sections are discussed after.

```java
/**
 * Basic pub/sub example
 * A TopicConnectionFactory object is retrieved from LDAP; this is used
 * to create a TopicConnection. The TopicConnection is used to create
 * a TopicSession, which creates two publishers and two subscribers.
 * Both publishers subscribe to a topic; both subscribers then receive.
 */
import javax.jms.*; // JMS interfaces
import javax.naming.*; // Used for JNDI lookup of
import javax.naming.directory.*; // administered objects
import java.io.*; // Java IO classes
import java.util.*; // Java Util classes

class PubSubSample {
    // using LDAP
    String icf = "com.sun.jndi.ldap.LdapCtxFactory"; // initial context factory
    String url = "ldap://server.company.com/o=company_us,c=us"; // url

    private String tcflookup = "cn=testTFC"; // TopicConnectionFactory (TFC) lookup
    private String tlookup = "cn=testT"; // topic lookup

    // Pub/Sub objects used by this program
    private TopicConnectionFactory fact = null;
    private Topic topic = null;

    public static void main(String args[])
    {
        // Initialise JNDI properties
        Hashtable env = new Hashtable();
        env.put( Context.INITIAL_CONTEXT_FACTORY, icf );
        env.put( Context.PROVIDER_URL, url );
        env.put( Context.REFERRAL, "true" );
```
Writing publish/subscribe application

Context ctx = null;
try {
    System.out.print( "Initialising JNDI... " );
    ctx = new InitialDirContext( env );
    System.out.println( "Done!" );
} catch ( NamingException nx ) {
    System.out.println( "ERROR: " + nx );
    System.exit(-1);
}

// Lookup TCF
try {
    System.out.print( "Obtaining TCF from JNDI... " );
    fact = (TopicConnectionFactory)ctx.lookup( tcfLookup );
    System.out.println( "Done!" );
} catch ( NamingException nx ) {
    System.out.println( "ERROR: " + nx );
    System.exit(-1);
}

// Lookup Topic
try {
    System.out.print( "Obtaining topic T from JNDI... " );
    topic = (Topic)ctx.lookup( tlookup );
    System.out.println( "Done!" );
} catch ( NamingException nx ) {
    System.out.println( "ERROR: " + nx );
    System.exit(-1);
}

try {
    ctx.close();
} catch ( NamingException nx ) {
    // Just ignore an exception on closing the context
}

try {
    // Create connection
    TopicConnection conn = fact.createTopicConnection();
    // Start connection
    conn.start();

    // Session
    TopicSession sess = conn.createTopicSession(false,
        Session.AUTO_ACKNOWLEDGE);

    // Create a topic dynamically
    Topic t = sess.createTopic("myTopic");

    // Publisher
    TopicPublisher pub = sess.createPublisher(t);
    // Subscriber
    TopicSubscriber sub = sess.createSubscriber(t);
    // Publisher
    TopicPublisher pubA = sess.createPublisher(topic);
    // Subscriber
    TopicSubscriber subA = sess.createSubscriber(topic);

    // Publish "Hello World"
    TextMessage hello = sess.createTextMessage();
    hello.setText("Hello World");
    pub.publish(hello);
    hello.setText("Hello World 2");
    pubA.publish(hello);
Write publish/subscribe application

// Receive message
TextMessage m = (TextMessage) sub.receive();
System.out.println("Message Text = " + m.getText());
m = (TextMessage) subA.receive();
System.out.println("Message Text = " + m.getText());

// Close publishers and subscribers
pub.close();
pubA.close();
sub.close();
subA.close();

// Close session and connection
sess.close();
conn.close();
System.exit(0);

} catch (JMSException je) {
    System.out.println("ERROR: " + je);
    System.out.println("LinkedException: " +
        je.getLinkedException());
    System.exit(-1);
}

Import required packages

The import statements for an application using WebSphere MQ classes for Java Message Service must include at least the following:
import javax.jms.*;       // JMS interfaces
import javax.naming.*;    // Used for JNDI lookup of
import javax.naming.directory.*;  // administered objects

Obtain or create JMS objects

The next step is to obtain or create a number of JMS objects:
1. Obtain a TopicConnectionFactory
2. Create a TopicConnection
3. Create a TopicSession
4. Obtain a Topic from JNDI
5. Create TopicPublishers and TopicSubscribers

Many of these processes are similar to those that are used for point-to-point, as shown in the following:

Obtain a TopicConnectionFactory

The preferred way to do this is to use JNDI lookup, to maintain portability of the application code. The following code initializes a JNDI context:

String icf = "com.sun.jndi.ldap.LdapCtxFactory"; // initial context factory
String url = "ldap://server.company.com/o=company_us,c=us"; // url

// Initialise JNDI properties
Java.util.Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, icf);
env.put(Context.PROVIDER_URL, url);
env.put(Context.REFERRAL, "throw");

Context ctx = null;
try {

}
System.out.print("Initialising JNDI...");
ctx = new InitialDirContext(env);
System.out.println("Done!");
}
catch (NamingException nx) {
   System.out.println("ERROR: " + nx);
   System.exit(-1);
}

Note: Change the icf and url variables to suit your installation and your JNDI service provider.

The properties required by JNDI initialization are in a Hashtable, which is passed to the InitialDirContext constructor. If this connection fails, an exception is thrown to indicate that the administered objects required later in the application are not available.

Obtain a TopicConnectionFactory using a lookup key that the administrator has defined:

// LOOKUP TCF
try {
   System.out.print("Obtaining TCF from JNDI...");
   fact = (TopicConnectionFactory)ctx.lookup(tcfLookup);
   System.out.println("Done!");
} catch (NamingException nx) {
   System.out.println("ERROR: " + nx);
   System.exit(-1);
}

If a JNDI namespace is not available, you can create a TopicConnectionFactory at runtime. You create a new com.ibm.mq.jms.MQTopicConnectionFactory as described in “Creating factories at runtime” on page 201.

Create a TopicConnection
This is created from the TopicConnectionFactory object. Connections are always initialized in a stop state and must be started with the following code:

// create connection
TopicConnection conn = fact.createTopicConnection();
//start connection
conn.start();

Create a TopicSession
This is created using the TopicConnection. This method takes two parameters: one to signify whether the session is transacted, and one to specify the acknowledgement mode:

TopicSession sess = conn.createTopicSession(false,
   Session.AUTO_ACKNOWLEDGE);

Obtain a Topic
This object can be obtained from JNDI, for use with TopicPublishers and TopicSubscribers that are created later. The following code retrieves a Topic:

Topic topic = null;
try {
   System.out.print("Obtaining topic T from JNDI...");
   topic = (Topic)ctx.lookup(tLookup);
   System.out.println("Done!");
}
catch ( NamingException nx ) {
    System.out.println( "ERROR: " + nx );
    System.exit(-1);
}

If a JNDI namespace is not available, you can create a Topic at runtime, as described in "Creating topics at runtime" on page 223.

The following code creates a Topic at runtime:
// topic
Topic t = sess.createTopic("myTopic");

Create consumers and producers of publications
Depending on the nature of the JMS client application that you write, a subscriber, a publisher, or both must be created. Use the createPublisher and createSubscriber methods as follows:
// publisher
TopicPublisher pub = sess.createPublisher(t);
// subscriber
TopicSubscriber sub = sess.createSubscriber(t);
// publisher
TopicPublisher pubA = sess.createPublisher(topic);
// subscriber
TopicSubscriber subA = sess.createSubscriber(topic);

Publish messages
The TopicPublisher object, pub, is used to publish messages, rather like a QueueSender is used in the point-to-point domain. The following fragment creates a TextMessage using the session, and then publishes the message:
// publish "hello world"
TextMessage hello = sess.createTextMessage();
hello.setText("Hello World");
pub.publish(hello);
hello.setText("Hello World 2");
pubA.publish(hello);

Receive subscriptions
Subscribers must be able to read the subscriptions that are delivered to them, as in the following code:
// receive message
TextMessage m = (TextMessage) sub.receive();
System.out.println("Message Text = " + m.getText());
m = (TextMessage) subA.receive();
System.out.println("Message Text = " + m.getText());

This fragment of code performs a get-with-wait, which means that the receive call blocks until a message is available. Alternative versions of the receive call are available (such as receiveNoWait). For details, see "TopicSubscriber" on page 440.

Close down unwanted resources
It is important to free up all the resources used by the application when it terminates. Use the close() method on objects that can be closed (publishers, subscribers, sessions, and connections):
// close publishers and subscribers
pub.close();
pubA.close();
sub.close();
subA.close();
sess.close();
// close session and connection
sess.close();
conn.close();

TopicConnectionFactory administered objects

In the example, the TopicConnectionFactory object is obtained from JNDI name space. The TopicConnectionFactory in this case is an administered object that has been created and administered using the JMSAdmin tool. Use this method of obtaining TopicConnectionFactory objects because it ensures code portability.

The TopicConnectionFactory in the example is testTCF in JMSAdmin. Create testTCF in JMSAdmin before running the application. You must also create a Topic in JMSAdmin; see "Topic administered objects."

To create a TopicConnectionFactory object, invoke the JMSAdmin tool, as described in "Invoking the administration tool" on page 41, and execute one of the following commands, depending on the type of connection you want to make to the broker:

**Bindings connection**

```
InitCtx> def tcf(testTCF) transport(bind)
```

or, because this is the default transport type for TopicConnectionFactory objects:

```
InitCtx> def tcf(testTCF)
```

This creates a TopicConnectionFactory with default settings for bindings transport, connecting to the default queue manager.

**Client connection**

```
InitCtx> def tcf(testTCF) transport(client)
```

This creates a TopicConnectionFactory with default settings for the client transport type, connecting to localhost, on port 1414, using channel SYSTEM.DEF.SVRCONN.

**Direct TCP/IP connection to WebSphere MQ Event Broker**

```
InitCtx> def tcf(testTCF) transport(direct)
```

This creates a TopicConnectionFactory to make direct connections to a WebSphere MQ Event Broker, connecting to localhost on port 1506.

Topic administered objects

In the example, one of the Topic objects has been obtained from JNDI name space. This Topic is an administered object that has been created and administered in the JMSAdmin tool. Use this method of obtaining Topic objects because it ensures code portability.

To run the example application above, create the Topic called testT in JMSAdmin before running the application.

To create a Topic object, invoke the JMSAdmin tool, as described in "Invoking the administration tool" on page 41, and execute one of the following commands, depending on the type of connection you want to make to the broker:

**Compatibility mode, or MQSeries Publish/Subscribe (SupportPac MA0C)**

```
```
Using topics

This section discusses the use of JMS Topic objects in WebSphere MQ classes for Java Message Service applications.

**Topic names**

This section describes the use of topic names within WebSphere MQ classes for Java Message Service.

**Note:** The JMS specification does not specify exact details about the use and maintenance of topic hierarchies. Therefore, this area can vary from one provider to the next.

Topic names in WebSphere MQ JMS are arranged in a tree-like hierarchy, an example of which is shown in Figure 3.

![Figure 3. WebSphere MQ classes for Java Message Service topic name hierarchy](image)

In a topic name, levels in the tree are separated by the `/` character. This means that the Signings node in Figure 3 is identified by the topic name:

```
Sport/Football/Spurs/Signings
```

A powerful feature of the topic system in WebSphere MQ classes for Java Message Service is the use of wildcards. These allow subscribers to subscribe to more than one topic at a time. Different brokers use different wildcard characters and different rules for their substitution. Use the broker version property of the topic (BROKERVER) to define which type of wildcards apply.

**Note:** The broker version of a topic must match the broker version of the topic connection factory you are using.

**Broker Version 1 wildcards**

The `*` wildcard matches zero or more characters; the `?` wildcard matches a single character.
Using topics

If a subscriber subscribes to the topic represented by the following topic name:

Sport/Football/*/Results

it receives publications on topics including:
- Sport/Football/Spurs/Results
- Sport/Football/Arsenal/Results

If the subscription topic is:

Sport/Football/Spurs/*

it receives publications on topics including:
- Sport/Football/Spurs/Results
- Sport/Football/Spurs/Signings

If the subscription topic is:

Sport/Football/*

it receives publications on topics including:
- Sport/Football/Arsenal/Results
- Sport/Football/Spurs/Results
- Sport/Football/Spurs/Signings

Broker Version 2 wildcards

The # wildcard matches multiple levels in a topic; the + wildcard matches a single level.

These wildcards can be used only to stand for complete levels within a topic; that is they can be preceded only by / or start-of-string, and they can be followed only by / or end-of-string.

If a subscriber subscribes to the topic represented by the following topic name:

Sport/Football/+/*Results

it receives publications on topics including:
- Sport/Football/Spurs/Results
- Sport/Football/Arsenal/Results

If a subscriber subscribes to the topic represented by the following topic name:

Sport/#/*Results

it receives publications on topics including:
- Sport/Football/Spurs/Results
- Sport/Football/Arsenal/Results

Although Sport/Football/Spur?/Results works with broker Version 1, there is no equivalent for broker Version 2, which does not support single character substitutions.

There is no need to administer the topic hierarchies that you use on the broker-side of your system explicitly. When the first publisher or subscriber on a given topic comes into existence, the broker automatically creates the state of the topics currently being published on, and subscribed to.

222 Using Java
Unicode characters are supported.

**Note:** A publisher cannot publish on a topic whose name contains wildcards.

### Creating topics at runtime

There are four ways to create Topic objects at runtime:

1. Construct a topic using the one-argument MQTopic constructor
2. Construct a topic using the default MQTopic constructor, and then call the setBaseTopicName(...) method
3. Use the session's createTopic(...) method
4. Use the session's createTemporaryTopic() method

#### Method 1: Using MQTopic(..)

This method requires a reference to the WebSphere MQ implementation of the JMS Topic interface, and therefore renders the code non-portable.

The constructor takes one argument, which must be a uniform resource identifier (URI). For WebSphere MQ classes for Java Message Service Topics, this must be of the form:

```
topic://TopicName[?property=value[&property=value]*]
```

For further details on URIs and the permitted name-value pairs, see "Sending a message" on page 204.

The following code creates a topic for non-persistent, priority 5 messages:

```java
// Create a Topic using the one-argument MQTopic constructor
String tSpec = "Sport/Football/Spurs/Results?persistence=1&priority=5";
Topic rtTopic = new MQTopic("topic:///" + tSpec);
```

#### Method 2: Using MQTopic(), then setBaseTopicName(..)

This method uses the default MQTopic constructor, and therefore renders the code non-portable.

After the object is created, set the baseTopicName property using the setBaseTopicName method, passing in the required topic name.

**Note:** The topic name used here is the non-URI form, and cannot include name-value pairs. Set these by using the set methods, as described in "Setting properties with the set method" on page 206. The following code uses this method to create a topic:

```java
// Create a Topic using the default MQTopic constructor
Topic rtTopic = new MQTopic();

// Set the object properties using the setter methods
((MQTopic)rtTopic).setBaseTopicName("Sport/Football/Spurs/Results");
((MQTopic)rtTopic).setPersistence(1);
((MQTopic)rtTopic).setPriority(5);
```

#### Method 3: Using session.createTopic(..)

You can also create a Topic object using the createTopic method of TopicSession, which takes a topic URI as follows:

```java
// Create a Topic using the session factory method
Topic rtTopic = session.createTopic( "topic://Sport/Football/Spurs/Results" );
```

Although the createTopic method is in the JMS specification, the format of the string argument is vendor-specific. Therefore, using this method might make your code non-portable.
Method 4: Using `session.createTemporaryTopic()`
A TemporaryTopic is a Topic that can be consumed only by subscribers that are created by the same TopicConnection. A TemporaryTopic is created as follows:

```java
// Create a TemporaryTopic using the session factory method
Topic rtTopic = session.createTemporaryTopic();
```

### Subscriber options

There are a number of different ways to use JMS subscribers. This section describes some examples of their use.

JMS provides two types of subscribers:

**Non-durable subscribers**

These subscribers receive messages on their chosen topic, only if the messages are published while the subscriber is active.

**Durable subscribers**

These subscribers receive all the messages published on a topic, including those that are published while the subscriber is inactive.

#### Creating non-durable subscribers

The subscriber created in "Create consumers and producers of publications" on page 218 is non-durable and is created with the following code:

```java
// Create a subscriber, subscribing on the given topic
TopicSubscriber sub = session.createSubscriber( topic );
```

#### Creating durable subscribers

Durable subscribers cannot be configured with a direct connection to WebSphere MQ Event Broker.

Creating a durable subscriber is very similar to creating a non-durable subscriber, but you must also provide a name that uniquely identifies the subscriber:

```java
// Create a durable subscriber, supplying a uniquely-identifying name
TopicSubscriber sub = session.createDurableSubscriber( topic, "D_SUB_000001" );
```

Non-durable subscribers automatically deregister themselves when their `close()` method is called (or when they fall out of scope). However, if you want to terminate a durable subscription, you must explicitly notify the system. To do this, use the session's `unsubscribe()` method and pass in the unique name that created the subscriber:

```java
// Unsubscribe the durable subscriber created above
session.unsubscribe( "D_SUB_000001" );
```

A durable subscriber is created at the queue manager specified in the MQTopicConnectionFactory queue manager parameter. If there is a subsequent attempt to create a durable subscriber with the same name at a different queue manager, a new and completely independent durable subscriber is returned.

#### Using message selectors

You can use message selectors to filter out messages that do not satisfy given criteria. For details about message selectors, see "Message selectors" on page 207. Message selectors are associated with a subscriber as follows:
// Associate a message selector with a non-durable subscriber
String selector = "company = 'IBM'"
TopicSubscriber sub = session.createSubscriber( topic, selector, false );

You can control whether the JMS client or the broker performs message filtering by
setting the MessageSelection property on the TopicConnectionFactory. If the broker
is capable of performing message selection, it is generally preferable to let the
broker do it because it reduces the number of messages sent to your client.
However, if the broker is very heavily loaded, it might be preferable to let the
client perform message selection instead.

Suppressing local publications
You can create a subscriber that ignores publications that are published on the
subscriber’s own connection. Set the third parameter of the createSubscriber call
to true, as follows:
// Create a non-durable subscriber with the noLocal option set
TopicSubscriber sub = session.createSubscriber( topic, null, true );

Combining the subscriber options
You can combine the subscriber variations, so that you can create a durable
subscriber that applies a selector and ignores local publications. The following code
fragment shows the use of the combined options:
// Create a durable, noLocal subscriber with a selector applied
String selector = "company = 'IBM'"
TopicSubscriber sub = session.createDurableSubscriber( topic, "D_SUB_00001",
selector, true );

Configuring the base subscriber queue
Subscriber queues cannot be configured for a direct connection to WebSphere
MQ Event Broker.

There are two ways in which you can configure subscribers:
• Multiple queue approach
  Each subscriber has an exclusive queue assigned to it, from which it retrieves all
  its messages. JMS creates a new queue for each subscriber. This is the only
  approach available with WebSphere MQ JMS V1.1.
• Shared queue approach
  A subscriber uses a shared queue, from which it, and other subscribers, retrieve
  their messages. This approach requires only one queue to serve multiple
  subscribers. This is the default approach used with WebSphere MQ JMS.

You can choose which approach to use, and configure which queues to use.

In general, the shared queue approach gives a modest performance advantage. For
systems with a high throughput, there are also large architectural and
administrative advantages, because of the significant reduction in the number of
queues required.

In some situations, there are still good reasons for using the multiple queue
approach:
• The theoretical physical capacity for message storage is greater.
  A WebSphere MQ queue cannot hold more than 640000 messages, and in the
  shared queue approach, this must be divided between all the subscribers that
  share the queue. This issue is more significant for durable subscribers, because
Subscriber options

the lifetime of a durable subscriber is usually much longer than that of a non-durable subscriber. Therefore, more messages might accumulate for a durable subscriber.

- External administration of subscription queues is easier.
   For certain application types, administrators might want to monitor the state and depth of particular subscriber queues. This task is much simpler when there is one to one mapping between a subscriber and a queue.

Default configuration
The default configuration uses the following shared subscription queues:
- SYSTEM.JMS.ND.SUBSCRIBER.QUEUE for non-durable subscriptions
- SYSTEM.JMS.D.SUBSCRIBER.QUEUE for durable subscriptions

These are created for you when you run the MQJMS_PSQ.MQSC script.

If required, you can specify alternative physical queues. You can also change the configuration to use the multiple queue approach.

Configuring non-durable subscribers
You can set the non-durable subscriber queue name property in either of the following ways:
- Use the WebSphere MQ JMS administration tool (for JNDI retrieved objects) to set the BROKERSUBQ property
- Use the setBrokerSubQueue() method in your program

For non-durable subscriptions, the queue name you provide should start with the following characters:
   SYSTEM.JMS.ND.

To select a shared queue approach, specify an explicit queue name, where the named queue is the one to use for the shared queue. The queue that you specify must already physically exist before you create the subscription.

To select the multiple queue approach, specify a queue name that ends with the * character. Subsequently, each subscriber that is created with this queue name creates an appropriate dynamic queue, for exclusive use by that particular subscriber. MQ JMS uses its own internal model queue to create such queues. Therefore, with the multiple queue approach, all required queues are created dynamically.

When you use the multiple queue approach, you cannot specify an explicit queue name. However, you can specify the queue prefix. This enables you to create different subscriber queue domains. For example, you could use:
   SYSTEM.JMS.ND.MYDOMAIN.*

The characters that precede the * character are used as the prefix, so that all dynamic queues that are associated with this subscription have queue names that start with SYSTEM.JMS.ND.MYDOMAIN.

Configuring durable subscribers
As discussed earlier, there might still be good reasons to use the multiple queue approach for durable subscriptions. Durable subscriptions are likely to have a longer life span, so it is possible that a large number of unretrieved messages could accumulate on the queue.
Therefore, the durable subscriber queue name property is set in the Topic object (that is, at a more manageable level than TopicConnectionFactory). This enables you to specify a number of different subscriber queue names, without needing to re-create multiple objects starting from the TopicConnectionFactory.

You can set the durable subscriber queue name in either of the following ways:

- Use the WebSphere MQ JMS administration tool (for JNDI retrieved objects) to set the BROKERDURSUBQ property
- Use the setBrokerDurSubQueue() method in your program:
  
  ```java
  // Set the MQTopic durable subscriber queue name using
  // the multi-queue approach
  sportsTopic.setBrokerDurSubQueue("SYSTEM.JMS.D.FOOTBALL.*");
  ```

Once the Topic object is initialized, it is passed into the TopicSession createDurableSubscriber() method to create the specified subscription:

  ```java
  // Create a durable subscriber using our earlier Topic
  TopicSubscriber sub = new session.createDurableSubscriber
  (sportsTopic, "D_SUB_SPORT_001");
  ```

For durable subscriptions, the queue name you provide must start with the following characters:

  SYSTEM.JMS.D.

To select a shared queue approach, specify an explicit queue name, where the named queue is the one to use for the shared queue. The queue that you specify must already physically exist before you create the subscription.

To select the multiple queue approach, specify a queue name that ends with the * character. Subsequently, each subscriber that is created with this queue name creates an appropriate dynamic queue, for exclusive use by that subscriber. MQ JMS uses its own internal model queue to create such queues. Therefore, with the multiple queue approach, all required queues are created dynamically.

When you use the multiple queue approach, you cannot specify an explicit queue name. However, you can specify the queue prefix. This enables you to create different subscriber queue domains. For example, you could use:

  SYSTEM.JMS.D.MYDOMAIN.*

The characters that precede the * character are used as the prefix, so that all dynamic queues that are associated with this subscription have queue names that start with SYSTEM.JMS.D.MYDOMAIN.

You cannot change the queue used by a durable subscriber. To do so, for example to move from the multiple queue approach to the single queue approach, first delete the old subscriber (using the unsubscribe() method) and create the subscription again from new. This removes any unconsumed messages on the durable subscription.

**Subscription stores**

There is no subscription store with a direct connection to WebSphere MQ Event Broker.

WebSphere MQ JMS maintains a persistent store of subscription information, used to resume durable subscriptions and cleanup after failed non-durable subscribers. This information can be managed by the publish/subscribe broker.
Subscriber options

See the README provided with WebSphere MQ JMS for information about suitable levels of queue manager and broker.

The choice of subscription store is based on the SUBSTORE property of the TopicConnectionFactory. This takes one of three values: QUEUE, BROKER, or MIGRATE.

SUBSTORE(QUEUE)
Subscription information is stored on SYSTEM.JMS.ADMIN.QUEUE and SYSTEM.JMS.PS.STATUS.QUEUE on the local queue manager.

WebSphere MQ JMS maintains an extra connection for a long-running transaction used to detect failed subscribers. There is a connection to each queue manager in use. In a busy system, this might cause the queue manager logs to fill up, resulting in errors from both the queue manager and its applications.

If you experience these problems, the system administrator can add extra log files or datasets to the queue manager. Alternatively, reduce the STATREFRESHINT property on the TopicConnectionFactory. This causes the long-running transaction to be refreshed more frequently, allowing the logs to reset themselves.

After a non-durable subscriber has failed:

- Information is left on the two SYSTEM.JMS queues, which allows a later JMS application to clean up after the failed subscriber. See “Subscriber cleanup utility” on page 230 for more information.
- Messages continue to be delivered to the subscriber until a later JMS application is executed.

SUBSTORE(QUEUE) is provided for compatibility with versions of MQSeries JMS.

SUBSTORE(BROKER)
Subscription information is stored by the publish/subscribe broker used by the application. This option requires recent levels of queue manager and publish/subscribe broker. See the README provided with WebSphere MQ JMS for information about suitable levels of queue manager and broker. This subscription store requires recent levels of both queue manager and publish/subscribe broker. It is designed to provide improved resilience.

When a non-durable subscriber fails, the subscription is deregistered from the broker as soon as possible. The broker adds a response to this deregistration onto the SYSTEM.JMS.REPORT.QUEUE, which is used to clean up after the failed subscriber. With SUBSTORE(BROKER), a separate cleanup thread is run regularly in the background of each JMS publish/subscribe application.

Cleanup is run once every 10 minutes by default, but you can change this using the CLEANUPINT property on the TopicConnectionFactory. To customize the actions performed by cleanup, use the CLEANUP property on the TopicConnectionFactory.

See “Subscriber cleanup utility” on page 230 for more information about the different behaviors of cleanup with SUBSTORE(BROKER).

SUBSTORE(MIGRATE)
MIGRATE is the default value for SUBSTORE.
This option dynamically selects the queue-based or broker-based subscription store based on the levels of queue manager and publish/subscribe broker installed. If both queue manager and broker are capable of supporting SUBSTORE(BROKER), this behaves as SUBSTORE(BROKER); otherwise it behaves as SUBSTORE(QUEUE). Additionally, SUBSTORE(MIGRATE) transfers durable subscription information from the queue-based subscription store to the broker-based store.

This provides an easy migration path from older versions of WebSphere MQ JMS, WebSphere MQ, and publish/subscribe broker. This migration occurs the first time the durable subscription is opened when both queue manager and broker are capable of supporting the broker-based subscription store. Only information relating to the subscription being opened is migrated; information relating to other subscriptions is left in the queue-based subscription store.

Migration and coexistence considerations
Except when SUBSTORE(MIGRATE) is used, the two subscription stores are entirely independent.

A durable subscription is created in the store specified by the TopicConnectionFactory. If there is a subsequent attempt to create a durable subscriber with the same name and ClientID but with the other subscription store, a new durable subscription is created.

When a connection uses SUBSTORE(MIGRATE), subscription information is automatically transferred from the queue-based subscription store to the broker-based subscription store when createDurableSubscriber() is called. If a durable subscription with matching name and ClientID already exists in the broker-based subscription store, this migration cannot complete and an exception is thrown from createDurableSubscriber().

Once a subscription has been migrated, it is important not to access the subscription from an application using an older version of WebSphere MQ JMS, or an application running with SUBSTORE(QUEUE). This would create a subscription in the queue-based subscription store, and prevent an application running with SUBSTORE(MIGRATE) from using the subscription.

To recover from this situation, either use SUBSTORE(BROKER) from your application or unsubscribe from the subscription with SUBSTORE(QUEUE).

For SUBSTORE(BROKER) to function, or for SUBSTORE(MIGRATE) to use the broker-based subscription store, suitable versions of both queue manager and broker need to be available to WebSphere MQ JMS. Refer to the README for information regarding suitable levels.

Solving publish/subscribe problems
This section describes some problems that can occur when you develop JMS client applications that use the publish/subscribe domain. It discusses problems that are specific to the publish/subscribe domain. Refer to "Handling errors" on page 209 and "Solving problems" on page 38 for more general troubleshooting guidance.
**Incomplete publish/subscribe close down**

It is important that JMS client applications surrender all external resources when they terminate. To do this, call the `close()` method on all objects that can be closed once they are no longer required. For the publish/subscribe domain, these objects are:

- TopicConnection
- TopicSession
- TopicPublisher
- TopicSubscriber

The WebSphere MQ classes for Java Message Service implementation eases this task by using a cascading close. With this process, a call to close on a TopicConnection results in calls to close on each of the TopicSessions it created. This in turn results in calls to close on all TopicSubscribers and TopicPublishers the sessions created.

To ensure the proper release of external resources, call `connection.close()` for each of the connections that an application creates.

There are some circumstances where this close procedure might not complete. These include:

- Loss of a WebSphere MQ client connection
- Unexpected application termination

In these circumstances, the `close()` is not called, and external resources remain open on the terminated application’s behalf. The main consequences of this are:

**Broker state inconsistency**

The WebSphere MQ Message Broker might contain registration information for subscribers and publishers that no longer exist. This means that the broker might continue forwarding messages to subscribers that will never receive them.

**Subscriber messages and queues remain**

Part of the subscriber deregistration procedure is the removal of subscriber messages. If appropriate, the underlying WebSphere MQ queue that was used to receive subscriptions is also removed. If normal closure has not occurred, these messages and queues remain. If there is broker state inconsistency, the queues continue to fill up with messages that will never be read.

Additionally, if the broker resides on a queue manager other than the application’s local queue manager, correct operation of WebSphere MQ JMS depends on the communication channels between the two queue managers. If these channels fail for any reason, problems such as the above can occur until the channels restart. When diagnosing problems relating to channels, be careful not to lose WebSphere MQ JMS control messages on the transmission queue.

**Subscriber cleanup utility**

To avoid the problems associated with non-graceful closure of subscriber objects, WebSphere MQ JMS includes a subscriber cleanup utility that attempts to detect any earlier WebSphere MQ JMS publish/subscribe problems that could have resulted from other applications. This utility runs transparently in the background and should not affect other WebSphere MQ JMS operations. If a large number of problems are detected against a given queue manager, you might see some performance degradation while resources are cleaned up.
Publish/subscribe problems

Note: Close all subscriber objects gracefully whenever possible, to avoid a buildup of subscriber problems.

The exact behavior of the utility depends on the subscription store in use:

**Subscriber cleanup with SUBSTORE(QUEUE)**

When using the queue-based subscription store, cleanup runs on a queue manager when the first TopicConnection to use that physical queue manager initializes.

If all the TopicConnections on a given queue manager close, when the next TopicConnection initializes for that queue manager, the utility runs again.

The cleanup utility uses information found on the SYSTEM.JMS.ADMIN.QUEUE and SYSTEM.JMS.PS.STATUS.QUEUE to detect previously failed subscribers. If any are found, it cleans up associated resources by deregistering the subscriber from the broker, and cleaning up any unconsumed messages or temporary queues associated with the subscription.

**Subscriber cleanup with SUBSTORE(BROKER)**

With the broker-based subscription store, cleanup runs regularly on a background thread while there is an open TopicConnection to a particular physical queue manager. One instance of the cleanup thread is created for each physical queue manager to which a TopicConnection exists within the JVM.

The cleanup utility uses information found on the SYSTEM.JMS.REPORT.QUEUE (typically responses from the publish/subscribe broker) to remove unconsumed messages and temporary queues associated with a failed subscriber. It can be a few seconds after the subscriber fails before the cleanup routine can remove the messages and queues.

Two properties on the TopicConnectionFactory control behavior of this cleanup thread: CLEANUP and CLEANUPINT. CLEANUPINT determines how often (in milliseconds) cleanup is executed against any given queue manager. CLEANUP takes one of four possible values:

**CLEANUP(SAFE)**

This is the default value.

The cleanup thread tries to remove unconsumed subscription messages or temporary queues for failed subscriptions. This mode of cleanup does not interfere with the operation of other JMS applications.

**CLEANUP(STRONG)**

The cleanup thread performs as CLEANUP(SAFE), but also clears the SYSTEM.JMS.REPORT.QUEUE of any unrecognized messages.

This mode of cleanup can interfere with the operation of JMS applications running with later versions of WebSphere MQ JMS. If multiple JMS applications are using the same queue manager, but using different versions of WebSphere MQ JMS, only clients using the most recent version of WebSphere MQ JMS must use this option.

**CLEANUP(NONE)**

In this special mode, no cleanup is performed, and no cleanup
thread exists. Additionally, if the application is using the single-queue approach, unconsumed messages can be left on the queue.

This option can be useful if the application is distant from the queue manager, and especially if it only publishes rather than subscribes. However, one application must clean up the queue manager to deal with any unconsumed messages. This can be a JMS application with CLEANUPSAFE or CLEANUPSTRONG, or the manual cleanup utility described in “Manual cleanup.”

CLEANUPASPROP
The style of cleanup to use is determined by the system property com.ibm.mq.jms.cleanup, which is queried at JVM startup.

This property can be set on the Java command-line using the -D option, to NONE, SAFE, or STRONG. Any other value causes an exception. If not set, the property defaults to SAFE.

This allows easy JVM-wide change to the cleanup level without updating every TopicConnectionFactory used by the system. This is useful for applications or application servers that use multiple TopicConnectionFactory objects.

Where multiple TopicConnections exist within a JVM against the same queue manager, but with differing values for CLEANUP and CLEANUPINT, the following rules are used to determine behavior:

1. A TopicConnection with CLEANUPNONE does not attempt to clean up immediately after its subscription has closed. However, if another TopicConnection is using SAFE or STRONG cleanup, the cleanup thread eventually cleans up after the subscription.

2. If any TopicConnection is using STRONG Cleanup, the cleanup thread operates at STRONG level. Otherwise, if any TopicConnection uses SAFE Cleanup, the cleanup thread operates at SAFE level. Otherwise, there is no cleanup thread.

3. The smallest value of CLEANUPINT for those TopicConnections with SAFE or STRONG Cleanup is used.

Manual cleanup
If you use the broker-based subscription store, you can operate cleanup manually from the command-line. The syntax for bindings attach is:

```
Cleanup [-m <qmgr>] [-r <interval>] [-t]
```

or, for client attach:

```
Cleanup -client [-m <qmgr>] -host <hostname> [-port <port>]
```

Where:
- qmgr, hostname, port, and channel determine connection settings for the queue manager to clean up.
- -r sets the interval between executions of cleanup, in minutes. If not set, cleanup is performed once.
- -t enables tracing, to the mqjms.trc file.
- SAFE, STRONG, FORCE, and NONDUR set the cleanup level, as follows:
SAFE and STRONG cleanup behave like the CLEANUP(SAFE) and CLEANUP(STRONG) options discussed in "Subscriber cleanup utility" on page 230.

FORCE cleanup behaves like STRONG Cleanup. However, STRONG cleanup leaves messages that could not be processed on the SYSTEM.JMS.REPORT.QUEUE; FORCE cleanup removes all messages even if it encounters an error during processing.

Warning
This is a dangerous option that can leave an inconsistent state between the queue manager and the broker. It cannot be run while any WebSphere MQ JMS publish/subscribe application is running against the queue manager; doing so causes the cleanup utility to abort.

NONDUR behaves like FORCE cleanup. After clearing the SYSTEM.JMS.REPORT.QUEUE, it attempts to remove any remaining unconsumed messages sent to non-durable subscribers. If the queue manager’s command server is running on any queue beginning SYSTEM.JMS.ND.*, messages are cleared and the queue itself might be deleted. Otherwise, only SYSTEM.JMS.ND.SUBSCRIBER.QUEUE and SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE are cleared of messages.

Cleanup from within a program
You can use a programming interface to the cleanup routines for use with SUBSTORE(BROKER), through the class com.ibm.mq.jms.Cleanup. Instances of this class have getter/setter methods for each of the connection properties; and also for the cleanup level and interval.

It exposes two methods:

```
cleanup()
```
Executes cleanup once

```
run()
```
Runs cleanup at intervals determined by the properties of the cleanup object

This class allows complete customization of publish/subscribe Cleanup, but it is intended for use by system administration programs rather than application programs.

For more details, refer to "Cleanup *" on page 308.

Handling broker reports
The WebSphere MQ JMS implementation uses report messages from the broker to confirm registration and deregistration commands. These reports are normally consumed by the WebSphere MQ classes for Java Message Service implementation, but under some error conditions, they might remain on the queue. These messages are sent to the SYSTEM.JMS.REPORT.QUEUE queue on the local queue manager.

A Java application, PSReportDump, is supplied with WebSphere MQ classes for Java Message Service, which dumps the contents of this queue in plain text format. The information can then be analyzed, either by you, or by IBM support staff. You can also use the application to clear the queue of messages after a problem is diagnosed or fixed.
The compiled form of the tool is installed in the `<MQ_JAVA_INSTALL_PATH>/bin` directory. To invoke the tool, change to this directory, then use the following command:

```
java PSReportDump [-m queueManager] [-clear]
```

where:

- `-m queueManager` - The name of the queue manager to use
- `-clear` - Clear the queue of messages after dumping its contents

**Attention:** Do not use this option if you are using the broker-based subscription store. Instead, run the manual cleanup utility at FORCE level.

Output is sent to the screen, or you can redirect it to a file.

### Other considerations

If a large number of JMS clients connect directly to a WebSphere MQ Event Broker broker on Windows, and the connections happen almost simultaneously, a `java.net.BindException address in use` exception might be thrown in response to a `TopicConnection` call. You can try to avoid this by catching the exception and retrying, or by pacing the connections.
Chapter 12. Writing WebSphere MQ JMS 1.1 applications

This chapter provides information to help you write WebSphere MQ JMS 1.1 applications. It covers information similar to that provided in Chapter 10, “Writing WebSphere MQ JMS applications,” on page 199 and Chapter 11, “Writing WebSphere MQ JMS publish/subscribe applications,” on page 213, but from a JMS 1.1 perspective.

The JMS 1.1 model

You can write a JMS application using two styles of messaging:

- Point-to-point
- Publish/subscribe

These styles of messaging are also referred to as messaging domains and you can combine both styles of messaging in one application.

With versions of JMS before JMS 1.1, programming for the point-to-point domain uses one set of interfaces and methods, and programming for the publish/subscribe domain uses another set. The two sets are similar, but separate. With JMS 1.1, you can use a common set of interfaces and methods that support both messaging domains. The common interfaces provide a domain independent view of each messaging domain. Table 17 lists the JMS 1.1 domain independent interfaces and their corresponding domain specific interfaces.

Table 17. The JMS 1.1 domain independent interfaces

<table>
<thead>
<tr>
<th>Domain independent interfaces</th>
<th>Domain specific interfaces for the point-to-point domain</th>
<th>Domain specific interfaces for the publish/subscribe domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionFactory</td>
<td>QueueConnectionFactory</td>
<td>TopicConnectionFactory</td>
</tr>
<tr>
<td>Connection</td>
<td>QueueConnection</td>
<td>TopicConnection</td>
</tr>
<tr>
<td>Destination</td>
<td>Queue</td>
<td>Topic</td>
</tr>
<tr>
<td>Session</td>
<td>QueueSession</td>
<td>TopicSession</td>
</tr>
<tr>
<td>MessageProducer</td>
<td>QueueSender</td>
<td>TopicPublisher</td>
</tr>
<tr>
<td>MessageConsumer</td>
<td>QueueReceiver</td>
<td>TopicSubscriber</td>
</tr>
<tr>
<td></td>
<td>QueueBrowser</td>
<td></td>
</tr>
</tbody>
</table>

JMS 1.1 retains all the domain specific interfaces in JMS 1.0.2b, and so existing applications can still use these interfaces. For new applications, however, consider using the JMS 1.1 domain independent interfaces.

In the WebSphere MQ JMS implementation of JMS 1.1, the administered objects are the following:

- ConnectionFactory
- Queue
- Topic

Destination is an abstract superclass of Queue and Topic, and so an instance of Destination is either a Queue or a Topic object. The domain independent interfaces...
treat a queue or a topic as a destination. The messaging domain for a MessageConsumer or a MessageProducer object is determined by whether the destination is a queue or a topic.

Building a connection

Connections are not created directly, but are built using a connection factory. ConnectionFactory objects can be stored in a JNDI namespace, insulating the JMS application from provider specific information. For information about how to create and store ConnectionFactory objects, see Chapter 5, “Using the WebSphere MQ JMS administration tool,” on page 41.

Retrieving a connection factory from JNDI

To retrieve a ConnectionFactory object from a JNDI namespace, you must first set up an initial context as shown in the following code:

```java
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
.java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
environment.put(Context.PROVIDER_URL, url);
Context ctx = new InitialDirContext(environment);
```

In this code:

- `icf` Defines a factory class for the initial context
- `url` Defines a context specific URL

For more details about using a JNDI namespace, see Sun’s JNDI documentation.

Note: Some combinations of the JNDI packages and LDAP service providers can result in an LDAP error 84. To resolve the problem, insert the following line before the call to InitialDirContext:

```java
environment.put(Context.REFERRAL, "throw");
```

After an initial context is obtained, you can retrieve a ConnectionFactory object from the JNDI namespace by using the lookup() method. The following code retrieves a ConnectionFactory object named CF from an LDAP based namespace:

```java
ConnectionFactory factory;
factory = (ConnectionFactory)ctx.lookup("cn=CF");
```

Using a connection factory to create a connection

You can use the createConnection() method on a ConnectionFactory object to create a Connection object, as shown in the following example:

```java
Connection connection;
connection = factory.createConnection();
```

Both QueueConnectionFactory and TopicConnectionFactory inherit the createConnection() method from ConnectionFactory. You can therefore use createConnection() to create a domain specific object, as shown in the following code:
Building a connection

```java
QueueConnectionFactory QCF;
Connection connection;
connection = QCF.createConnection();
```

This code creates a QueueConnection object. An application can now perform a domain independent operation on this object, or an operation that is applicable only to the point-to-point domain. If the application attempts to perform an operation that is applicable only to the publish/subscribe domain, an IllegalStateException is thrown with the message MQJMS1112: JMS 1.1 Invalid operation for a domain specific object. This is because the connection was created from a domain specific connection factory.

### Creating a connection factory at runtime

If a JNDI namespace is not available, it is possible to create a ConnectionFactory object at runtime. However, using this method reduces the portability of a JMS application because the application must then include references to WebSphere MQ specific classes.

The following code creates a ConnectionFactory object with all the default settings:

```java
factory = new com.ibm.mq.jms.MQConnectionFactory();
```

The default transport type is bindings. You can change the transport type for a connection factory by using the setTransportType() method. Here are some examples:

```java
fact.setTransportType(JMSC.MQJMS_TP_BINDINGS_MQ);  // Bindings mode
fact.setTransportType(JMSC.MQJMS_TP_CLIENT_MQ_TCP);  // Client mode
fact.setTransportType(JMSC.MQJMS_TP_DIRECT_TCP);  // Direct TCP/IP mode
```

For information about transport types in each of the specific messaging domains, see "Choosing client or bindings transport" on page 202 for the point-to-point domain, and "TopicConnectionFactory administered objects" on page 220 for the publish/subscribe domain.

Note that you cannot use the point-to-point style of messaging if the transport type is direct. If an application uses Connection and Session objects that are created from a ConnectionFactory object whose transport type is direct, the application can perform only publish/subscribe operations.

A ConnectionFactory object has the same properties as those of a QueueConnectionFactory object and a TopicConnectionFactory object combined. However, certain combinations of property settings are not valid for a ConnectionFactory object. See "Properties" on page 49 for more details.

### Starting the connection

The JMS specification states that a connection is created in the stopped state. Until the connection starts, a message consumer that is associated with the connection cannot receive any messages. To start the connection, issue the following command:

```java
connection.start();
```

### Specifying a range of ports for client connections

If a JMS application attempts to connect to a WebSphere MQ queue manager in client mode, a firewall might allow only those connections that originate from specified ports or a range of ports. In this situation, you can use the
Building a connection

LOCALADDRESS property of a ConnectionFactory, QueueConnectionFactory, or TopicConnectionFactory object to specify a port, or a range of points, that the application can bind to.

You can set the LOCALADDRESS property by using the WebSphere MQ JMS administration tool, or by calling the setLocalAddress() method in a JMS application. Here is an example of setting the property from within an application:

mqConnectionFactory.setLocalAddress("9.20.0.1(2000,3000)");

When the application connects to a queue manager subsequently, the application binds to a local IP address and port number in the range 9.20.0.1(2000) to 9.20.0.1(3000).

Connection errors might occur if you restrict the range of ports. If an error occurs, a JMSException is thrown with an embedded MQException that contains the WebSphere MQ reason code, MQRC_Q_MGR_NOT_AVAILABLE. An error might occur if all the ports in the specified range are in use, or if the LOCALADDRESS property contains an IP address, host name, or port number that is not valid; a negative port number, for example.

Because the WebSphere MQ JMS client might create connections other than those required by an application, always consider specifying a range of ports. In general, every Session created by an application requires one port and the WebSphere MQ JMS client might require three additional ports. If a connection error does occur, increase the range of ports.

JMS connection pooling, which is used by default, might have an effect on the speed at which ports can be reused. As a result, a connection error might occur while ports are being freed.

Obtaining a session

After a connection is made, use the createSession() method on the Connection object to obtain a session. The method has two parameters:

1. A boolean parameter that determines whether the session is transacted or non-transacted.
2. A parameter that determines the acknowledge mode.

The simplest case is obtaining a non-transacted session with an acknowledge mode of AUTO_ACKNOWLEDGE, as shown in the following code:

```
Session session;

boolean transacted = false;
session = connection.createSession(transacted, Session.AUTO_ACKNOWLEDGE);
```

Note: A connection is thread safe, but sessions (and the objects that are created from them) are not. The recommended practice for multithreaded applications is to use a separate session for each thread.
Destinations

The Destination interface is the abstract superclass of Queue and Topic. In the WebSphere MQ JMS implementation of JMS 1.1, Queue and Topic objects encapsulate addresses in WebSphere MQ and the broker. For example, a Queue object encapsulates the name of a WebSphere MQ queue.

For information about using Queue objects in the point-to-point domain, see "Sending a message" on page 204 and, for information about using Topic objects in the publish/subscribe domain, see "Using topics" on page 221. The following is a overview from a domain independent perspective.

Queue and Topic objects are retrieved from a JNDI namespace in the following way:

```java
Queue ioQueue;
ioQueue = (Queue)ctx.lookup( qLookup );
...
Topic ioTopic;
ioTopic = (Topic)ctx.lookup( tLookup );
```

The WebSphere MQ JMS implementation of Queue and Topic interfaces are in the com.ibm.mq.jms.MQQueue and com.ibm.jms.MQTopic classes respectively. These classes contain properties that control the behavior of WebSphere MQ and the broker but, in many cases, it is possible to use the default values. As well as being able to administer Queue and Topic objects in a JNDI namespace, JMS defines a standard way of specifying a destination at runtime that minimizes the WebSphere MQ specific code in the application. This mechanism uses the Session.createQueue() and Session.createTopic() methods, which take a string parameter that specifies the destination. The string is still in a provider specific format, but this approach is more flexible than referring directly to the provider classes.

WebSphere MQ JMS accepts two forms for the string parameter of createQueue():

1. The first is the name of a WebSphere MQ queue:
   ```java
   public static final String QUEUE = "SYSTEM.DEFAULT.LOCAL.QUEUE";
   ...
   ioQueue = session.createQueue( QUEUE );
   ```

2. The second, and more powerful, form is a uniform resource identifier (URI).
   This form allows you to specify a remote queue, which is a queue on a queue manager other than the one to which you are connected. It also allows you to set the other properties of a com.ibm.mq.jms.MQQueue object.

   The URI for a queue begins with the sequence queue://, followed by the name of the queue manager on which the queue resides. This is followed by a further forward slash (/), the name of the queue, and, optionally, a list of name-value pairs that set the remaining queue properties. For example, the URI equivalent of the previous example is the following:
   ```java
   ioQueue = session.createQueue("queue://SYSTEM.DEFAULT.LOCAL.QUEUE");
   ```

   The name of the queue manager is omitted. This is interpreted to mean as the queue manager to which the owning Connection object is connected at the time when the Queue object is used.
Destinations

**Note:** When sending a message to a cluster, leave the queue manager field in the Queue object blank. This enables an MQOPEN call to be performed in BIND_NOT_FIXED mode, which allows the queue manager to be determined. Otherwise an exception is returned reporting that the Queue object cannot be found. This applies when using JNDI or defining a queue at runtime.

WebSphere MQ JMS accepts a topic URI for the string parameter of `createTopic()`, as shown in the following example:

```java
Topic topic = session.createTopic( "topic://Sport/Football/Spurs/Results" );
```

Although the `createTopic()` method is in the JMS specification, the format of the string argument is provider specific. Therefore, using this method can make your code non-portable.

Other ways of creating a Topic object at runtime are as follows:

**Using `MQTopic(..)`**
This way requires a reference to the WebSphere MQ JMS implementation of the `Topic` interface, and therefore renders the code non-portable.

The constructor takes one argument, which must be a URI. For a WebSphere MQ JMS topic, this must be of the form:

```java
topic://TopicName?[property=value[&property=value]*]
```

For example, the following code creates a topic for nonpersistent messages with a priority of 5:

```java
// Create a Topic using the one-argument MQTopic constructor
String tSpec = "Sport/Football/Spurs/Results?persistence=1&priority=5";
Topic rtTopic = new MQTopic( "topic://" + tSpec );
```

**Using `MQTopic()`, then `setBaseTopicName(..)`**
This way uses the default `MQTopic` constructor, and therefore renders the code non-portable. Here is an example:

```java
// Create a Topic using the default MQTopic constructor
Topic rtTopic = new MQTopic();
```

```java
// Set the object properties using the setter methods
((MQTopic)rtTopic).setBaseTopicName( "Sport/Football/Spurs/Results" );
((MQTopic)rtTopic).setPersistence(1);
((MQTopic)rtTopic).setPriority(5);
```

**Using `session.createTemporaryTopic()`**
A temporary topic is created by a session, and only message consumers created by the same session can consume messages from the topic. A TemporaryTopic object is created as follows:

```java
// Create a TemporaryTopic using the session factory method
Topic rtTopic = session.createTemporaryTopic();
```

Sending a message

An application sends messages using a MessageProducer object. A MessageProducer object is normally created for a specific destination so that all messages sent using that message producer are sent to the same destination. The destination is specified using either a Queue or a Topic object. Queue and Topic objects can be created at runtime, or built and stored in a JNDI namespace, as described in "Destinations" on page 239.
Sending a message

After a Queue or a Topic object is obtained, an application can pass the object to the createProducer() method to create a MessageProducer object, as shown in the following example:

```java
MessageProducer messageProducer = session.createProducer(ioDestination);
```

The parameter ioDestination can be either a Queue or a Topic object.

The application can then use the send() method on the MessageProducer object to send messages. Here is an example:

```java
messageProducer.send(outMessage);
```

You can use the send() method to send messages in either messaging domain. The nature of the destination determines the actual domain used. However, TopicPublisher, the sub-interface for MessageProducer that is specific to the publish/subscribe domain, uses a publish() method instead.

An application can create a MessageProducer object with no specified destination. In this case, the application must specify the destination in the send() method.

Message types

JMS provides several message types, each of which embodies some knowledge of its content. To avoid referring to the provider specific class names for the message types, methods for creating messages are provided on a Session object.

For example, a text message can be created in the following manner:

```java
System.out.println( "Creating a TextMessage" );
TextMessage outMessage = session.createTextMessage();
System.out.println("Adding Text");
outMessage.setText(outString);
```

Here are the message types you can use:

- BytesMessage
- MapMessage
- ObjectMessage
- StreamMessage
- TextMessage

Details of these types are in Chapter 15, “JMS interfaces and classes,” on page 295.

Receiving a message

An application receives messages using a MessageConsumer object. The application creates a MessageConsumer object by using the createConsumer() method on a Session object. This method has a destination parameter that defines where the messages are received from. See [“Destinations” on page 239] for details of how to create a destination, which is either a Queue or a Topic object.

In the point-to-point domain, the following code creates a MessageConsumer object and then uses the object to receive a message:

```java
MessageConsumer messageConsumer = session.createConsumer(ioQueue);
Message inMessage = messageConsumer.receive(1000);
```

The parameter on the receive() call is a timeout in milliseconds. This parameter defines how long the method must wait if no message is available immediately.
Receiving a message

You can omit this parameter; in which case, the call blocks until a suitable message arrives. If you do not want any delay, use the `receiveNoWait()` method.

The `receive()` methods return a message of the appropriate type. For example, suppose a text message is put on a queue. When the message is received, the object that is returned is an instance of `TextMessage`.

To extract the content from the body of the message, it is necessary to cast from the generic `Message` class (which is the declared return type of the `receive()` methods) to the more specific subclass, such as `TextMessage`. If the received message type is not known, you can use the `instanceof` operator to determine which type it is. It is good practice always to test the message class before casting so that unexpected errors can be handled gracefully.

The following code uses the `instanceof` operator and shows how to extract the content of a text message:

```java
if (inMessage instanceof TextMessage) {
    String replyString = ((TextMessage) inMessage).getText();
    
} else {
    // Print error message if Message was not a TextMessage.
    System.out.println("Reply message was not a TextMessage");
}
```

JMS provides two types of message consumer:

**Nondurable message consumer**

A nondurable message consumer receives messages from its chosen destination only if the messages are available while the consumer is active.

In the point-to-point domain, whether a consumer receives messages that are sent while the consumer is inactive depends on how WebSphere MQ is configured to support that consumer. In the publish/subscribe domain, a consumer does not receive messages that are sent while the consumer is inactive.

**Durable topic subscriber**

A durable topic consumer receives all the messages sent to a destination, including those sent while the consumer is inactive.

The following sections describe how to create a durable topic subscriber, and how to configure WebSphere MQ and the broker to support either type of message consumer.

**Creating durable topic subscribers**

You cannot create a durable topic subscriber if the transport type is direct.

Durable topic subscribers are used when an application needs to receive messages that are published even while the application is inactive. Creating a durable topic subscriber is very similar to creating a nondurable message consumer, but you must also provide a name that uniquely identifies the subscription, as in the following example:

```java
// Create a durable subscriber, supplying a uniquely-identifying name
TopicSubscriber sub = session.createDurableSubscriber( topic, "D_SUB_000001" );
```
Receiving a message

A durable topic subscriber is created for the queue manager specified by the QMANAGER property of the MQTopicConnectionFactory object. If there is a subsequent attempt to create a durable topic subscriber with the same name for a different queue manager, a new and completely independent durable topic subscriber is returned.

Nondurable message consumers in the publish/subscribe domain automatically deregister themselves when their close() method is called, or when they fall out of scope. However, if you want to terminate a durable subscription, you must explicitly notify the broker. To do this, use the unsubscribe() method of the session and pass in the name that uniquely identifies the subscription:

```java
// Unsubscribe the durable subscriber created above
session.unsubscribe( "D_SUB_000001" );
```

**Message selectors**

JMS allows an application to specify that only messages that satisfy certain criteria are returned by successive receive() calls. When creating a MessageConsumer object, you can provide a string that contains an SQL (Structured Query Language) expression, which determines which messages are retrieved. This SQL expression is called a *selector*. The selector can refer to fields in the JMS message header as well as fields in the message properties (these are effectively application defined header fields). Details of the header field names, as well as the syntax for an SQL selector, are in Chapter 13, “JMS messages,” on page 257.

The following example shows how to select messages based on a user defined property called myProp:

```java
messageConsumer = session.createConsumer(ioQueue, "myProp = 'blue'");
```

**Note:** The JMS specification does not permit the selector associated with a message consumer to be changed. After a message consumer is created, the selector is fixed for the lifetime of that consumer. This means that, if you require different selectors, you must create new message consumers.

In the publish/subscribe domain, you can control whether the JMS client or the broker performs message filtering by setting the MSGSELECTION property on the ConnectionFactory object. If the broker is capable of performing message selection, it is generally preferable to let the broker do it because it reduces the amount of work done by the client. However, if the broker is very heavily loaded, it might be preferable to let the client perform message selection instead.

**Suppressing local publications**

You can create a message consumer that ignores publications published on the consumer's own connection. To do this, set the third parameter on the createConsumer() call to true, as shown in the following example:

```java
// Create a nondurable message consumer with the nolocal option set
MessageConsumer con = session.createConsumer( topic, null, true );
```

The example that follows shows how to create a durable topic subscriber that applies a selector and ignores local publications:

```java
// Create a durable, nolocal subscriber with a selector applied
String selector = "company = 'IBM'";
TopicSubscriber sub = session.createDurableSubscriber( topic, "D_SUB_000001",
selector, true );
```
Configuring the consumer queue

You cannot configure a consumer queue if the transport type is direct.

In the publish/subscribe messaging domain, you can configure message consumers in two ways:

- The multiple queue approach.
  Each consumer has its own exclusive queue and retrieves all its messages from this queue. JMS creates a new queue for each consumer.
- The shared queue approach.
  Each consumer retrieves its messages from a queue that is shared with other consumers. This approach requires only one queue to serve multiple consumers. It is the default approach used with WebSphere MQ JMS.

You can choose which approach to use, and configure which queues to use.

In general, there is a modest performance advantage if you use the shared queue approach. For systems with a high throughput, there are also large system management and administrative advantages because of the significant reduction in the number of queues required.

In some situations, however, there are good reasons for using the multiple queue approach:

- In theory, you can store more messages.
  A WebSphere MQ queue cannot hold more than 640000 messages and so, in the shared queue approach, the total number of messages for all the message consumers that share the queue cannot exceed this limit. This issue is more significant for durable topic subscribers, because the lifetime of a durable topic subscriber is usually much longer than that of a nondurable message consumer. Therefore, more messages might accumulate for a durable subscriber.
- The WebSphere MQ administration of consumer queues is easier.
  For certain applications, an administrator might want to monitor the state and depth of particular consumer queues. This task is much simpler when each consumer has its own queue.

Default configuration

The default WebSphere MQ JMS configuration for the publish/subscribe domain uses the following shared consumer queues:

- SYSTEM.JMS.ND.SUBSCRIBER.QUEUE for nondurable message consumers
- SYSTEM.JMS.D.SUBSCRIBER.QUEUE for durable topic subscribers

These are created for you when you run the MQJMS_PSQ.MQSC script.

If required, you can specify alternative WebSphere MQ queues. You can also change the configuration to use the multiple queue approach.

Configuring nondurable message consumers

You can specify the name of the consumer queue for nondurable message consumers in either of the following ways:

- Use the WebSphere MQ JMS administration tool to set the BROKERSUBQ property.
- Use the setBrokerSubQueue() method in your application.

The queue name you provide must start with the following characters:

SYSTEM.JMS.ND.
Receiving a message

To use the shared queue approach, specify the complete name of the shared queue. The queue must exist before you can create a subscription.

To use the multiple queue approach, specify a queue name that ends with an asterisk (*). Subsequently, when an application creates a nondurable message consumer specifying this queue name, WebSphere MQ JMS creates a temporary dynamic queue for exclusive use by that consumer. With the multiple queue approach, therefore, all the required queues are created dynamically.

If you use the multiple queue approach, you cannot specify the complete name of a queue, only a prefix. This allows you to create different domains of consumer queues. For example, you can use:

```
SYSTEM.JMS.ND.MYDOMAIN.*
```

The characters that precede the asterisk (*) are used as the prefix, so that all dynamic queues for nondurable message consumers specifying this prefix have queue names that start with SYSTEM.JMS.ND.MYDOMAIN.

Configuring durable topic subscribers

As stated previously, there might still be good reasons to use the multiple queue approach for durable topic subscribers. Durable topic subscribers are likely to have a longer life span, and so it is possible for a large number of messages for a durable subscriber to accumulate on a queue.

The name of the consumer queue for a durable topic subscriber is a property of a Topic object. This allows you to specify a number of different consumer queue names without having to create multiple objects starting from a ConnectionFactory object.

You can specify the name of the consumer queue for durable topic subscribers in either of the following ways:

- Use the WebSphere MQ JMS administration tool to set the BROKERDURSUBQ property.
- Use the setBrokerDurSubQueue() method in your application.

The queue name you provide must start with the following characters:

```
SYSTEM.JMS.D.
```

To use the shared queue approach, specify the complete name of the shared queue. The queue must exist before you can create a subscription.

To use the multiple queue approach, specify a queue name that ends with an asterisk (*). Subsequently, when an application creates a durable topic subscriber specifying this queue name, WebSphere MQ JMS creates a permanent dynamic queue for exclusive use by that subscriber. With the multiple queue approach, therefore, all the required queues are created dynamically.

Here is an example of using the multiple queue approach:

```
// Set the MQTopic durable subscriber queue name using
// the multi-queue approach
sportsTopic.setBrokerDurSubQueue("SYSTEM.JMS.D.FOOTBALL.*");
```

After the Topic object is initialized, it can be passed into the createDurableSubscriber() method of a Session object to create a durable topic subscriber:
Receiving a message

// Create a durable subscriber using our earlier Topic
TopicSubscriber sub = session.createDurableSubscriber(sportsTopic,
        "D_SUB_SPORT_001");

If you use the multiple queue approach, you cannot specify the complete name of
a queue, only a prefix. This allows you to create different domains of consumer
queues. For example, you can use:

    SYSTEM.JMS.D.MYDOMAIN.*

The characters that precede the asterisk (*) are used as the prefix, so that all
dynamic queues for durable topic subscribers specifying this prefix have queue
names that start with SYSTEM.JMS.D.MYDOMAIN.

You cannot change the consumer queue of a durable topic subscriber. If, for
example, you want to move from the multiple queue approach to the single queue
approach, you must first delete the old subscriber using the unsubscribe() method
and then create a new subscriber. Deleting the old subscriber also deletes any
unconsumed messages for that subscriber.

Subscription stores

A subscription store is not used if the transport type is direct.

When an application creates a message consumer in the publish/subscribe domain,
information about the subscription is created at the same time. WebSphere MQ
JMS maintains a persistent store of subscription information called a subscription
store. A subscription store is used to reopen durable topic subscribers and to clean
up after a nondurable message consumer fails. A subscription store can be
managed by the local queue manager or by the publish/subscribe broker.

The SUBSTORE property of a ConnectionFactory object determines the location of
a subscription store. SUBSTORE has three possible values:

SUBSTORE(QUEUE)

Subscription information is stored in the queues,
SYSTEM.JMS.ADMIN.QUEUE and SYSTEM.JMS.PS.STATUS.QUEUE, on
the local queue manager.

WebSphere MQ JMS maintains an extra connection to each queue manager
used by applications. This connection is used to detect an application that
fails and to clean up after the application. In a busy system, this might
cause the queue manager logs to fill up resulting in errors from both the
queue manager and the applications connected to it.

If you experience these problems, the system administrator can add extra
log files or data sets to the queue manager. Alternatively, you can reduce
the STATREFRESHINT property of the ConnectionFactory object. This
causes the long running transaction to be refreshed more frequently
allowing the logs to reset themselves.

After a nondurable message consumer fails, the following occurs:

• Subscription information related to the failed consumer remains on the
two queues implementing the subscription store. This information can be
removed by a cleanup utility supplied with WebSphere MQ JMS. See
"Consumer cleanup utility for the publish/subscribe domain" on page
248 for more information.

• Messages continue to be delivered to the consumer until the cleanup
utility runs.
This option is provided for compatibility with versions of MQSeries JMS.

**SUBSTORE(BROKER)**
Subscription information is stored by the publish/subscribe broker used by the application, not in WebSphere MQ queues. This means that, if a JMS client fails, the broker can clean up the resources associated with the JMS client without having to wait for the JMS client to reconnect. See the README provided with WebSphere MQ JMS for information about which release levels of WebSphere MQ and the broker support this option.

If a nondurable message consumer fails, the subscription is de-registered from the broker as soon as possible. In response to the de-registration, the broker puts a report message on the queue, SYSTEM.JMS.REPORT.QUEUE. This message is used to clean up after the failed consumer.

If you use this option, a separate cleanup thread is run in the background. By default, the cleanup utility runs once every 10 minutes, but you can change this interval by setting the CLEANUPINT property of the ConnectionFactory object. To customize the actions performed by the cleanup utility, use the CLEANUP property of the ConnectionFactory object. For more information about how the cleanup utility works, see “Consumer cleanup utility for the publish/subscribe domain” on page 248.

**SUBSTORE(MIGRATE)**
This is the default value.

This option dynamically selects a queue based or a broker based subscription store depending on the release levels of WebSphere MQ and the publish/subscribe broker that are installed. If both WebSphere MQ and the broker are capable of supporting the SUBSTORE(BROKER) option, this option behaves like the SUBSTORE(BROKER) option; otherwise, it behaves like the SUBSTORE(QUEUE) option.

If this option behaves like the SUBSTORE(BROKER) option, the option additionally migrates durable subscription information from the queue based subscription store to the broker based store. This migration occurs the first time a durable subscription is opened when both WebSphere MQ and the broker are capable of supporting a broker based subscription store. Only information related to the subscription being opened is migrated. Information related to other subscriptions is left in the queue based subscription store. This option therefore provides an easy migration path from older versions of WebSphere MQ JMS, WebSphere MQ, and the publish/subscribe broker.

**Migration and coexistence considerations**
Except when the SUBSTORE(MIGRATE) option is used, a queue based subscription store and a broker based subscription store are entirely independent.

A durable subscription is created in the subscription store specified by the ConnectionFactory object. If there is a subsequent attempt to create a durable subscription with the same name and ClientID, but with the other subscription store, a new durable subscription is created.

When a connection uses the SUBSTORE(MIGRATE) option, subscription information is automatically migrated from the queue based subscription store to the broker based subscription store when the application calls the createDurableSubscriber() method. If a durable subscription with a matching name and ClientID already exists in the broker based subscription store, the migration cannot complete and an exception is thrown by the createDurableSubscriber() call.
Receiving a message

After a subscription is migrated, it is important not to access the subscription from an application using an older version of WebSphere MQ JMS, or from an application running with the SUBSTORE(QUEUE) option. Doing either of these creates a subscription in the queue based subscription store and prevents an application running with the SUBSTORE(MIGRATE) option from using the subscription.

To recover from this situation if it occurs, run your application with the SUBSTORE(BROKER) option, or unsubscribe from the subscription that is held in the queue based subscription store.

Asynchronous delivery

An application can call the MessageConsumer.receive() method to receive messages. As an alternative, an application can register a method that is called automatically when a suitable message is available. This is called asynchronous delivery of messages. The following code illustrates the mechanism:

```java
import javax.jms.*;

public class MyClass implements MessageListener
{
    // The method that will be called by JMS when a message
    // is available.
    public void onMessage(Message message)
    {
        System.out.println("message is "+message);

        // application specific processing here
        .
        .
        .
    }
    .
    .
    .
    // In Main program (possibly of some other class)
    MyClass listener = new MyClass();
    messageConsumer.setMessageListener(listener);
    // main program can now continue with other application specific
    // behavior.
}
```

**Note:** Using asynchronous delivery with a message consumer marks the entire session as using asynchronous delivery. An application cannot call the receive() methods of a message consumer if the message consumer is associated with a session that is using asynchronous delivery.

Consumer cleanup utility for the publish/subscribe domain

To avoid the problems associated with message consumer objects in the publish/subscribe domain not closing gracefully, WebSphere MQ JMS supplies a consumer cleanup utility that attempts to clean up the resources associated with a consumer that has failed. This utility runs in the background and does not affect other WebSphere MQ JMS operations. If the utility detects a large number of problems associated with a given queue manager, you might see some performance degradation while resources are being cleaned up.

**Note:** Whenever possible, close all message consumer objects gracefully to avoid an accumulation of these types of problems.
If applications use the domain independent classes, the cleanup utility is invoked only if the applications perform publish/subscribe operations, such as creating a Topic object, or creating a MessageConsumer object with a Topic object retrieved from a JNDI namespace. This is to prevent the cleanup utility from being invoked in an environment in which applications are performing only point-to-point operations.

The exact behavior of the cleanup utility depends on where the subscription store is located:

**Queue based subscription store**
For a queue based subscription store, the cleanup utility runs against a queue manager when the first Connection object to use that queue manager initializes. If all the Connection objects that use a given queue manager close, the utility runs again only when the next Connection object to use that queue manager initializes.

The cleanup utility uses the information in the queues, SYSTEM.JMS.ADMIN.QUEUE and SYSTEM.JMS.PS.STATUS.QUEUE, to detect nondurable message consumers that have failed previously. If it finds a failed consumer, the utility cleans up the resources associated with the consumer by de-registering the consumer from the broker and deleting its consumer queue, provided it is not a shared queue, and any unconsumed messages on the queue.

**Broker based subscription store**
For a broker based subscription store, the cleanup utility runs at regular intervals on a background thread while there is at least one Connection object that uses a given queue manager. One cleanup thread is created for each queue manager for which a Connection object exists within the JVM.

The cleanup utility uses information in the queue, SYSTEM.JMS.REPORT.QUEUE (the messages in this queue are typically report messages from the publish/subscribe broker), to perform any necessary cleanup. This might involve deleting consumer queues and unconsumed messages that are no longer required.

Two properties of a ConnectionFactory object control the behavior of the cleanup thread: CLEANUPINT and CLEANUP. CLEANUPINT determines how often, in milliseconds, the cleanup utility is run against any given queue manager. CLEANUP has four possible values:

**CLEANUP(SAFE)**
This is the default value.

The cleanup thread tries to delete any consumer queues and unconsumed messages that are no longer required. This mode of cleanup does not interfere with the operation of other JMS applications.

**CLEANUP(STRONG)**
The cleanup thread performs like the CLEANUP(SAFE) option, but it also deletes any messages on the queue, SYSTEM.JMS.REPORT.QUEUE, that it does not recognize.

This mode of cleanup can interfere with the operation of JMS applications running with later versions of WebSphere MQ JMS. If multiple JMS applications are using the same queue manager, but
Publish/subscribe domain consumer cleanup utility

using different versions of WebSphere MQ JMS, only applications using the most recent version of WebSphere MQ JMS must use this option.

CLEANUP(NONE)
In this special mode, no cleanup is performed, and consumer queues and unconsumed messages that are no longer required are not deleted.

This option can be useful if the application and the queue manager are on a different systems, especially if the application only sends messages and does not receive them. At some time, however, cleanup must be initiated to delete consumer queues and unconsumed messages that are no longer required. This can be done by a JMS application that uses a ConnectionFactory object with the property CLEANUP(SAFE) or CLEANUP(STRONG), or by using the manual cleanup utility, which is described in Manual cleanup.

CLEANUP(ASPROP)
The mode of cleanup is determined by the system property com.ibm.mq.jms.cleanup, which is queried when the JVM starts.

This property can be set on the Java command line by using the -D option. Its value can be SAFE, STRONG, or NONE. Any other value causes an exception. If the property not set, its value defaults to SAFE.

This option allows you to change the mode of cleanup within an entire JVM without having to update every ConnectionFactory object. This is useful for applications or application servers that use multiple ConnectionFactory objects.

If multiple Connection objects for the same queue manager exist within a JVM, but the Connection objects use different values for the CLEANUPINT and CLEANUP properties, the following rules determine the behavior of the cleanup utility:

1. If a Connection object using the CLEANUP(NONE) option fails, cleanup does not run. The cleanup thread eventually runs, however, if another Connection object is using the CLEANUP(SAFE) or CLEANUP(STRONG) option.

2. If any Connection object is using the CLEANUP(STRONG) option, the cleanup thread operates in STRONG mode. Otherwise, if any Connection object is using the CLEANUP(SAFE) option, the cleanup thread operates in SAFE mode. Otherwise, there is no cleanup thread.

3. The cleanup utility runs at intervals determined by the smallest value of the CLEANUPINT property of those Connections that are using the CLEANUP(SAFE) or CLEANUP(STRONG) option.

Manual cleanup
If you use a broker based subscription store, you can operate the cleanup utility manually from the command line. Here is the syntax of the command:

For a bindings connection:

    Cleanup [-m <qmgr>] [-r <interval>]
            [SAFE | STRONG | FORCE | NODUR] [-t]

For a client connection:
Publish/subscribe domain consumer cleanup utility

`Cleanup -client [-m <qmgr>] -h <hostname> [-p <port>]`
`[-c <channel>] [-r <interval>]`
`[-t SAFE | STRONG | FORCE | NONDUR] [-i]`

The parameters of the command are as follows:

- `qmgr`, `hostname`, `port`, and `channel` enable the cleanup utility to connect to a queue manager.
- `-r` sets the interval, in minutes, between each run of the cleanup utility. If the parameter is not set, the cleanup utility runs once only.
- `-t` enables tracing. The output is sent to the file `mqjms.trc`.
- `SAFE`, `STRONG`, `FORCE`, or `NONDUR` sets the cleanup level as follows:
  - `SAFE` and `STRONG` behave like the `CLEANUP(SAFE)` and `CLEANUP(STRONG)` modes discussed in "Consumer cleanup utility for the publish/subscribe domain" on page 248.
  - `FORCE` behaves like `STRONG` mode. But, whereas `STRONG` mode leaves any messages that cannot be processed on the queue, `SYSTEM.JMS.REPORT.QUEUE`, `FORCE` mode deletes all the messages even if it encounters an error during processing.

```
<table>
<thead>
<tr>
<th>Warning</th>
</tr>
</thead>
</table>
| This is a dangerous mode that can leave an inconsistent state between the queue manager and the broker. You cannot run the cleanup utility in this mode while any WebSphere MQ JMS publish/subscribe applications are connected to the queue manager. If you try to do so, the cleanup utility ends.

- `NONDUR` behaves like `FORCE` mode but, in addition, this mode deletes all the messages on queues whose names begin with the characters `SYSTEM.JMS.ND`. To do this successfully, the command server of the queue manager must be running.
```

**Cleanup from within a program**

You can use a programming interface to invoke the cleanup utility that is used with a broker based subscription service. Instances of the class `com.ibm.mq.jms.Cleanup` have getter and setter methods for each of the properties that are used to connect to a queue manager, and also for the cleanup level and cleanup interval. It exposes two additional methods:

- `cleanup()` executes the cleanup utility once only.
- `run()` runs cleanup at intervals determined by the cleanup interval property.

This class allows complete customization of the publish/subscribe cleanup utility, but it is intended for use by system administration programs rather than application programs.

For more details, see "Cleanup *" on page 308.
Closing down

Garbage collection alone cannot release all WebSphere MQ resources in a timely manner, especially if an application creates many short lived JMS objects at the session level or lower. It is therefore important for an application to call the appropriate close() method to close a Connection, Session, MessageConsumer, or MessageProducer object when it is no longer required.

Java Virtual Machine hangs at shutdown

If an application using WebSphere MQ JMS finishes without calling Connection.close(), some JVMs appear to hang. If this problems occurs, you can end the JVM by entering Ctrl-C. To avoid the problem in the future, consider modifying the application to include a call to Connection.close().

Handling errors

Any runtime errors in a JMS application are reported by exceptions. The majority of JMS methods throw a JMSException to indicate an error. It is good programming practice to catch these exceptions and display them on a suitable output device.

A JMSException can contain a further exception embedded within it. For JMS, this can be a valuable way to pass important information about the error from the underlying transport. In the case of WebSphere MQ JMS, an MQException is thrown in WebSphere MQ base Java whenever an error occurs in WebSphere MQ, and this exception is usually included as the embedded exception in a JMSException.

The implementation of JMSException does not include the embedded exception in the output of its toString() method. Therefore, you must check explicitly for an embedded exception and print it out, as shown in the following example:

```java
try {
    // code which may throw a JMSException
} catch (JMSException je) {
    System.err.println("caught "+je);
    Exception e = je.getLinkedException();
    if (e != null) {
        System.err.println("linked exception: "+e);
    }
}
```

Exception listener

For asynchronous message delivery, the application code cannot catch exceptions raised by failures to receive messages. This is because the application code does not make explicit calls to receive() methods. To cope with this situation, you can register an ExceptionListener, which is an instance of a class that implements the onException() method. When a serious error occurs, this method is called with the JMSException passed as its only parameter. Further details are in Sun’s JMS documentation.

Handling broker reports

The WebSphere MQ JMS implementation uses report messages from the broker to confirm whether registration and de-registration requests have been successful. These report messages are sent to the queue, SYSTEM.JMS.REPORT.QUEUE, on
Handling errors

the local queue manager and are normally consumed by the WebSphere MQ JMS.
Under some error conditions, however, they might remain on the queue.

WebSphere MQ JMS supplies a Java application, PSReportDump, which dumps the
contents of the queue, SYSTEM.JMS.REPORT.QUEUE, in plain text format. The
information in the dump can be analyzed by you or by IBM support staff. You can
also use the application to delete all the messages in the queue after a problem is
diagnosed or fixed.

The compiled form of the application is in the <MQ_JAVA_INSTALL_PATH>/bin
directory. To start the application, change to this directory and use the following
command:

```java
java PSReportDump [-m queueManager] [-clear]
```

where:

- `-m queueManager`
  - is the name of the queue manager to use
- `-clear`
  - causes all the messages on the queue to be deleted after their contents have
    been dumped

**Attention:** Do not use this option if you are using a broker based
subscription store. Instead, run the manual cleanup utility in FORCE mode.

The output from the application is sent to the screen, or you can redirect it to a
file.

Other considerations

If a large number of JMS clients connect directly to a WebSphere MQ Event Broker
broker on Windows, and the connections happen almost simultaneously, a
java.net.BindException address in use exception might be thrown in response to a
request to connect to the broker. You can try to avoid this by catching the
exception and retrying, or by pacing the connections.

User exits

WebSphere MQ JMS allows you to implement send, receive, and security exits
using interfaces supplied by WebSphere MQ base Java. For WebSphere MQ JMS,
ensure that your exit has a constructor that takes a single string argument. See the
description of exit related set methods in [Table 14 on page 202](#) and "Property
dependencies" on page 56.

Using Secure Sockets Layer (SSL)

WebSphere MQ base Java client applications and WebSphere MQ JMS connections
using TRANSPORT(CLIENT) support Secure Sockets Layer (SSL) encryption. SSL
provides communication encryption, authentication, and message integrity. It is
typically used to secure communications between any two peers on the Internet or
within an intranet.

WebSphere MQ classes for Java uses Java Secure Socket Extension (JSSE) to handle
SSL encryption, and so requires a JSSE provider. J2SE v1.4 JVMs have a JSSE
provider built in. Details of how to manage and store certificates can vary from
provider to provider. For information about this, refer to your JSSE provider’s
documentation.
Using SSL

This section assumes that your JSSE provider is correctly installed and configured, and that suitable certificates have been installed and made available to your JSSE provider.

SSL administrative properties

This section introduces the SSL administrative properties, as follows:

- "SSLCIPHERSUITE object property"
- "SSLPEERNAME object property"
- "SSLCERTSTORES object property" on page 255
- "SSLSocketFactory object property" on page 256

SSLCIPHERSUITE object property

To enable SSL encryption on a ConnectionFactory object, use JMSAdmin to set the SSLCIPHERSUITE property to a CipherSuite supported by your JSSE provider. This must match the CipherSpec set on the target channel. However, CipherSuites are distinct from CipherSpecs and so have different names. Appendix H, "SSL CipherSuites supported by WebSphere MQ," on page 487 contains a table mapping the CipherSpecs supported by WebSphere MQ to their equivalent CipherSuites as known to JSSE. Additionally, the named CipherSuite must be supported by your JSSE provider. For more information about CipherSpecs and CipherSuites with WebSphere MQ, see the WebSphere MQ Security book.

For example, to set up a ConnectionFactory object that can be used to create a connection over an SSL enabled MQI channel with a CipherSpec of RC4_MD5_EXPORT, issue the following command to JMSAdmin:

`ALTER CF(my.cf) SSLCIPHERSUITE(SSL_RSA_EXPORT_WITH_RC4_40_MD5)`

This can also be set from an application, using the setSSLCipherSuite() method on an MQConnectionFactory object.

For convenience, if a CipherSpec is specified on the SSLCIPHERSUITE property, JMSAdmin attempts to map the CipherSpec to an appropriate CipherSuite and issues a warning. This attempt to map is not made if the property is specified by an application.

SSLPEERNAME object property

A JMS application can ensure that it connects to the correct queue manager by specifying a distinguished name (DN) pattern. The connection succeeds only if the queue manager presents a DN that matches the pattern. For more details of the format of this pattern, refer to WebSphere MQ Security or the WebSphere MQ Script (MQSC) Command Reference.

The DN is set using the SSLPEERNAME property of a ConnectionFactory object. For example, the following JMSAdmin command sets a ConnectionFactory object to expect the queue manager to identify itself with a Common Name beginning with the characters QMGR., and with at least two Organizational Unit names, the first of which must be IBM and the second WEBSHERE:

`ALTER CF(my.cf) SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSHERE)`

Checking is not case sensitive and semicolons can be used in place of commas. This can also be set from an application using the setSSLPeerName() method on an MQConnectionFactory object. If this property is not set, no checking is performed on the DistinguishedName supplied by the queue manager. This property is ignored if no CipherSuite is set.
SSL_CERTSTORES object property

It is common to use a certificate revocation list (CRL) to identify certificates that are no longer trusted. CRLs are typically hosted on LDAP servers. JMS allows an LDAP server to be specified for CRL checking under Java 2 v1.4 or later. The following JMSAdmin example directs JMS to use a CRL hosted on an LDAP server named crl1.ibm.com:

```bash
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com)
```

**Note:** To use a CertStore successfully with a CRL hosted on an LDAP server, make sure that your Java Software Development Kit (SDK) is compatible with the CRL. Some SDKs require that the CRL conforms to RFC 2587, which defines a schema for LDAP v2. Most LDAP v3 servers use RFC 2256 instead.

If your LDAP server is not running on the default port of 389, the port can be specified by appending a colon (:) and the port number to the host name. If the certificate presented by the queue manager is present in the CRL hosted on crl1.ibm.com, the connection does not complete. To avoid a single point of failure, JMS allows multiple LDAP servers to be supplied by supplying a list of LDAP servers delimited by the space character. Here is an example:

```bash
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com ldap://crl2.ibm.com)
```

When multiple LDAP servers are specified, JMS tries each one in turn until it finds a server with which it can successfully verify the queue manager’s certificate. Each server must contain identical information.

A string in this format can be supplied by an application on the MQConnectionFactory.setSSLCertStores() method. Alternatively, the application can create one or more java.security.cert.CertStore objects, place these in a suitable Collection object, and supply this Collection object to the setSSLCertStores() method. In this way, the application can customize CRL checking. Refer to your JSSE documentation for details on constructing and using CertStore objects.

The certificate presented by the queue manager when a connection is being set up is validated as follows:

1. The first CertStore object in the Collection identified by sslCertStores is used to identify a CRL server.
2. An attempt is made to contact the CRL server.
3. If the attempt is successful, the server is searched for a match for the certificate.
   a. If the certificate is found to be revoked, the search process is over and the connection request fails with reason code MQRC_SSL_CERTIFICATE_REVOKED.
   b. If the certificate is not found, the search process is over and the connection is allowed to proceed.
4. If the attempt to contact the server is unsuccessful, the next CertStore object is used to identify a CRL server and the process repeats from step 2.

If this was the last CertStore in the Collection, or if the Collection contains no CertStore objects, the search process has failed and the connection request fails with reason code MQRC_SSL_CERT_STORE_ERROR.

The Collection object determines the order in which CertStores are used.

If your application uses `setSSLCertStores()` to set a Collection of CertStore objects, the MQConnectionFactory can no longer be bound into a JNDI namespace. Attempting to do so causes an exception. If the sslCertStores property is not set, no
revocation checking is performed on the certificate provided by the queue manager. This property is ignored if no CipherSuite is set.

**SSLSocketFactory object property**

You might want to customize other aspects of the SSL connection for an application. For example, you might want to initialize cryptographic hardware or change the KeyStore and TrustStore in use. To do this, the application must first create a javax.net.ssl.SSLSocketFactory object that is customized accordingly. Refer to your JSSE documentation for information on how to do this, as the customizable features vary from provider to provider. After a suitable SSLSocketFactory object is obtained, use the MQConnectionFactory.setSSLSocketFactory() method to configure JMS to use the customized SSLSocketFactory object.

If your application uses the setSSLSocketFactory() method to set a customized SSLSocketFactory object, the MQConnectionFactory object can no longer be bound into a JNDI namespace. Attempting to do so causes an exception. If this property is not set, the default SSLSocketFactory object is used. Refer to your JSSE documentation for details on the behavior of the default SSLSocketFactory object. This property is ignored if no CipherSuite is set.

**Important:** Do not assume that the use of the SSL properties ensures security when a ConnectionFactory object is retrieved from a JNDI namespace that is not itself secure. Specifically, the standard LDAP implementation of JNDI is not secure. An attacker can imitate the LDAP server, misleading a JMS application into connecting to the wrong server without noticing. With suitable security arrangements in place, other implementations of JNDI (such as the fscontext implementation) are secure.
Chapter 13. JMS messages

JMS messages are composed of the following parts:

**Header**

All messages support the same set of header fields. Header fields contain values that are used by both clients and providers to identify and route messages.

**Properties**

Each message contains a built-in facility to support application-defined property values. Properties provide an efficient mechanism to filter application-defined messages.

**Body**

JMS defines several types of message body that cover the majority of messaging styles currently in use.

JMS defines five types of message body:

- **Stream**
  A stream of Java primitive values. It is filled and read sequentially.

- **Map**
  A set of name-value pairs, where names are strings and values are Java primitive types. The entries can be accessed sequentially or randomly by name. The order of the entries is undefined.

- **Text**
  A message containing a java.util.String.

- **Object**
  A message that contains a serializable Java object

- **Bytes**
  A stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format.

The JMSCorrelationID header field is used to link one message with another. It typically links a reply message with its requesting message. JMSCorrelationID can hold a provider-specific message ID, an application-specific String, or a provider-native byte[] value.

**Message selectors**

A message contains a built-in facility to support application-defined property values. In effect, this provides a mechanism to add application-specific header fields to a message. Properties allow an application, using message selectors, to have a JMS provider select or filter messages on its behalf, using application-specific criteria. Application-defined properties must obey the following rules:

- Property names must obey the rules for a message selector identifier.
- Property values can be boolean, byte, short, int, long, float, double, and string.
- The JMSX and JMS_ name prefixes are reserved.

Property values are set before sending a message. When a client receives a message, the message properties are read-only. If a client attempts to set properties at this point, a MessageNotWriteableException is thrown. If clearProperties is called, the properties can now be both read from, and written to.
Message selectors

A property value might duplicate a value in a message’s body. JMS does not define a policy for what should or should not be made into a property. However, application developers must be aware that JMS providers probably handle data in a message’s body more efficiently than data in a message’s properties. For best performance, applications must use message properties only when they need to customize a message’s header. The primary reason for doing this is to support customized message selection.

A JMS message selector allows a client to specify the messages that it is interested in by using the message header. Only messages whose headers match the selector are delivered.

Message selectors cannot refer to message body values.

A message selector matches a message when the selector evaluates to true when the message’s header field and property values are substituted for their corresponding identifiers in the selector.

A message selector is a String, whose syntax is based on a subset of the SQL92 conditional expression syntax. The order in which a message selector is evaluated is from left to right within a precedence level. You can use parentheses to change this order. Predefined selector literals and operator names are written here in upper case; however, they are not case-sensitive.

A selector can contain:

- **Literals**
  - A string literal is enclosed in single quotes. A doubled single quote represents a single quote. Examples are ‘literal’ and ’literal’’. Like Java string literals, these use the Unicode character encoding.
  - An exact numeric literal is a numeric value without a decimal point, such as 57, -957, and +62. Numbers in the range of Java long are supported.
  - An approximate numeric literal is a numeric value in scientific notation, such as 7E3 or -57.9E2, or a numeric value with a decimal, such as 7., -95.7, or +6.2. Numbers in the range of Java double are supported.
  - The boolean literals TRUE and FALSE.

- **Identifiers**:
  - An identifier is an unlimited length sequence of Java letters and Java digits, the first of which must be a Java letter. A letter is any character for which the method Character.isJavaLetter returns true. This includes _ and $. A letter or digit is any character for which the method Character.isJavaLetterOrDigit returns true.
  - Identifiers cannot be the names NULL, TRUE, or FALSE.
  - Identifiers cannot be NOT, AND, OR, BETWEEN, LIKE, IN, or IS.
  - Identifiers are either header field references or property references.
  - Identifiers are case-sensitive.
  - Message header field references are restricted to:
    - JMSDeliveryMode
    - JMSPriority
    - JMSMessageID
    - JMSTimestamp
    - JMSCorrelationID
Message selectors

- JMSType
  JMSMessageID, JMSTimestamp, JMSCorrelationID, and JMSType values can be null, and if so, are treated as a NULL value.
  - Any name beginning with JMSX is a JMS-defined property name.
  - Any name beginning with JMS_ is a provider-specific property name.
  - Any name that does not begin with JMS is an application-specific property name. If there is a reference to a property that does not exist in a message, its value is NULL. If it does exist, its value is the corresponding property value.

- White space is the same as it is defined for Java: space, horizontal tab, form feed, and line terminator.
- Expressions:
  - A selector is a conditional expression. A selector that evaluates to true matches; a selector that evaluates to false or unknown does not match.
  - Arithmetic expressions are composed of themselves, arithmetic operations, identifiers (whose value is treated as a numeric literal), and numeric literals.
  - Conditional expressions are composed of themselves, comparison operations, and logical operations.

- Standard bracketing (), to set the order in which expressions are evaluated, is supported.
- Logical operators in precedence order: NOT, AND, OR.
- Comparison operators: =, >, >=, <, <=, <> (not equal).
  - Only values of the same type can be compared. One exception is that it is valid to compare exact numeric values and approximate numeric values. (The type conversion required is defined by the rules of Java numeric promotion.) If there is an attempt to compare different types, the selector is always false.
  - String and boolean comparison is restricted to = and <>. Two strings are equal only if they contain the same sequence of characters.

- Arithmetic operators in precedence order:
  - +, - unary.
  - *, /, multiplication, and division.
  - +, -, addition, and subtraction.
  - Arithmetic operations on a NULL value are not supported. If they are attempted, the complete selector is always false.
  - Arithmetic operations must use Java numeric promotion.
- arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 and arithmetic-expr3 comparison operator:
  - Age BETWEEN 15 and 19 is equivalent to age >= 15 AND age <= 19.
  - Age NOT BETWEEN 15 and 19 is equivalent to age < 15 OR age > 19.
  - If any of the expressions of a BETWEEN operation are NULL, the value of the operation is false. If any of the expressions of a NOT BETWEEN operation are NULL, the value of the operation is true.

- identifier [NOT] IN (string-literal1, string-literal2,...) comparison operator where identifier has a String or NULL value.
  - Country IN ('UK', 'US', 'France') is true for 'UK' and false for 'Peru'. It is equivalent to the expression (Country = 'UK') OR (Country = 'US') OR (Country = 'France').
  - Country NOT IN ('UK', 'US', 'France') is false for 'UK' and true for 'Peru'. It is equivalent to the expression NOT ((Country = 'UK') OR (Country = 'US') OR (Country = 'France')).
Message selectors

- If the identifier of an IN or NOT IN operation is NULL, the value of the operation is unknown.
- identifier [NOT] LIKE pattern-value [ESCAPE escape-character] comparison operator, where identifier has a string value. pattern-value is a string literal, where _ stands for any single character and % stands for any sequence of characters (including the empty sequence). All other characters stand for themselves. The optional escape-character is a single character string literal, whose character is used to escape the special meaning of the _ and % in pattern-value.
  - phone LIKE '12%3' is true for 123 and 12993 and false for 1234.
  - word LIKE 'l_se' is true for lose and false for loose.
  - underscored LIKE '\_%' ESCAPE '\' is true for _foo and false for bar.
  - phone NOT LIKE '12%3' is false for 123 and 12993 and true for 1234.
  - If the identifier of a LIKE or NOT LIKE operation is NULL, the value of the operation is unknown.
- identifier IS NULL comparison operator tests for a null header field value, or a missing property value.
  - prop_name IS NULL.
- identifier IS NOT NULL comparison operator tests for the existence of a non-null header field value or a property value.
  - prop_name IS NOT NULL.

The following message selector selects messages with a message type of car, color of blue, and weight greater than 2500 lbs:

"JMSType = 'car' AND color = 'blue' AND weight > 2500"

As noted above, property values can be NULL. The evaluation of selector expressions that contain NULL values is defined by SQL 92 NULL semantics. The following is a brief description of these semantics:

- SQL treats a NULL value as unknown.
- Comparison or arithmetic with an unknown value always yields an unknown value.
- The IS NULL and IS NOT NULL operators convert an unknown value into the respective TRUE and FALSE values.

Although SQL supports fixed decimal comparison and arithmetic, JMS message selectors do not. This is why exact numeric literals are restricted to those without a decimal. It is also why there are numerics with a decimal as an alternate representation for an approximate numeric value.

SQL comments are not supported.
Mapping JMS messages onto WebSphere MQ messages

This section describes how the JMS message structure that is described in the first part of this chapter is mapped onto a WebSphere MQ message. It is of interest to programmers who want to transmit messages between JMS and traditional WebSphere MQ applications. It is also of interest to people who want to manipulate messages transmitted between two JMS applications, for example, in a message broker implementation.

This section does not apply when you use a direct connection to WebSphere MQ Event Broker.

WebSphere MQ messages are composed of three components:
- The WebSphere MQ Message Descriptor (MQMD)
- A WebSphere MQ MQRFH2 header
- The message body.

The MQRFH2 is optional, and its inclusion in an outgoing message is governed by a flag in the JMS Destination class. You can set this flag using the WebSphere MQ JMS administration tool. Because the MQRFH2 carries JMS-specific information, always include it in the message when the sender knows that the receiving destination is a JMS application. Normally, omit the MQRFH2 when sending a message directly to a non-JMS application. This is because such an application does not expect an MQRFH2 in its WebSphere MQ message. Figure 4 shows how the structure of a JMS message is transformed to a WebSphere MQ message and back again:

```
Figure 4. How messages are transformed between JMS and WebSphere MQ using the MQRFH2 header
```

The structures are transformed in two ways:

**Mapping**
Where the MQMD includes a field that is equivalent to the JMS field, the JMS field is mapped onto the MQMD field. Additional MQMD fields are exposed as JMS properties, because a JMS application might need to get or set these fields when communicating with a non-JMS application.

**Copying**
Where there is no MQMD equivalent, a JMS header field or property is passed, possibly transformed, as a field inside the MQRFH2.
The MQRFH2 header

This section describes the MQRFH Version 2 header, which carries JMS-specific data that is associated with the message content. The MQRFH2 Version 2 is an extensible header, and can also carry additional information that is not directly associated with JMS. However, this section covers only its use by JMS.

There are two parts of the header, a fixed portion and a variable portion.

Fixed portion

The fixed portion is modelled on the standard WebSphere MQ header pattern and consists of the following fields:

**StrucId (MQCHAR4)**
Structure identifier.
Must be MQRFH_STRUC_ID (value: “RFH ”) (initial value).
MQRFH_STRUC_ID_ARRAY (value: “R”, “F”, “H”, “”) is also defined in the usual way.

**Version (MQLONG)**
Structure version number.
Must be MQRFH_VERSION_2 (value: 2) (initial value).

**StrucLength (MQLONG)**
Total length of MQRFH2, including the NameValueData fields.
The value set into StrucLength must be a multiple of 4 (the data in the NameValueData fields can be padded with space characters to achieve this).

**Encoding (MQLONG)**
Data encoding.
Encoding of any numeric data in the portion of the message following the MQRFH2 (the next header, or the message data following this header).

**CodedCharSetId (MQLONG)**
Coded character set identifier.
Representation of any character data in the portion of the message following the MQRFH2 (the next header, or the message data following this header).

**Format (MQCHAR8)**
Format name.
Format name for the portion of the message following the MQRFH2.

**Flags (MQLONG)**
Flags.
MQRFH_NO_FLAGS =0. No flags set.

**NameValueCCSID (MQLONG)**
The coded character set identifier (CCSID) for the NameValueData character strings contained in this header. The NameValueData can be coded in a character set that differs from the other character strings that are contained in the header (StrucID and Format).
If the NameValueCCSID is a 2-byte Unicode CCSID (1200, 13488, or 17584), the byte order of the Unicode is the same as the byte ordering of the numeric fields in the MQRFH2. (For example, Version, StrucLength, and NameValueCCSID itself.)

The NameValueCCSID takes values from the following table:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>UCS2 open-ended</td>
</tr>
<tr>
<td>1208</td>
<td>UTF8</td>
</tr>
<tr>
<td>13488</td>
<td>UCS2 2.0 subset</td>
</tr>
<tr>
<td>17584</td>
<td>UCS2 2.1 subset (includes Euro symbol)</td>
</tr>
</tbody>
</table>

Variable portion

The variable portion follows the fixed portion. The variable portion contains a variable number of MQRFH2 folders. Each folder contains a variable number of elements or properties. Folders group together related properties. The MQRFH2 headers created by JMS can contain up to three folders:

The `<mcd>` folder

This contains properties that describe the shape or format of the message. For example, the `Msd` property identifies the message as being Text, Bytes, Stream, Map, Object, or null. This folder is always present in a JMS MQRFH2.

The `<jms>` folder

This is used to transport JMS header fields, and JMSX properties that cannot be fully expressed in the MQMD. This folder is always present in a JMS MQRFH2.

The `<usr>` folder

This is used to transport any application-defined properties associated with the message. This folder is only present if the application has set some application-defined properties.

Table 19 shows a full list of property names.

<table>
<thead>
<tr>
<th>JMS field name</th>
<th>Java type</th>
<th>MQRFH2 folder name</th>
<th>Property name</th>
<th>Type/values</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMSDestination</td>
<td>Destination</td>
<td>jms</td>
<td>Dst</td>
<td>string</td>
</tr>
<tr>
<td>JMSExpiration</td>
<td>long</td>
<td>jms</td>
<td>Exp</td>
<td>i8</td>
</tr>
<tr>
<td>JMSPriority</td>
<td>int</td>
<td>jms</td>
<td>Pri</td>
<td>i4</td>
</tr>
<tr>
<td>JMSDeliveryMode</td>
<td>int</td>
<td>jms</td>
<td>Dlv</td>
<td>i4</td>
</tr>
<tr>
<td>JMSCorrelationID</td>
<td>String</td>
<td>jms</td>
<td>Cid</td>
<td>string</td>
</tr>
<tr>
<td>JMSReplyTo</td>
<td>Destination</td>
<td>jms</td>
<td>Rto</td>
<td>string</td>
</tr>
<tr>
<td>JMSTimestamp</td>
<td>long</td>
<td>jms</td>
<td>Tms</td>
<td>i8</td>
</tr>
<tr>
<td>JMSType</td>
<td>String</td>
<td>mcd</td>
<td>Type, Set, Fmt</td>
<td>string</td>
</tr>
<tr>
<td>JMSXGroupID</td>
<td>String</td>
<td>jms</td>
<td>Gid</td>
<td>string</td>
</tr>
<tr>
<td>JMSXGroupSeq</td>
<td>int</td>
<td>jms</td>
<td>Seq</td>
<td>i4</td>
</tr>
<tr>
<td>xxx (user defined)</td>
<td>Any</td>
<td>usr</td>
<td>xxx</td>
<td>any</td>
</tr>
</tbody>
</table>
Table 19. MQRFH2 folders and properties used by JMS (continued)

<table>
<thead>
<tr>
<th>JMS field name</th>
<th>Java type</th>
<th>MQRFH2 folder name</th>
<th>Property name</th>
<th>Type/values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>mcd</td>
<td>Msd</td>
<td>jms_none</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>jms_text</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>jms_bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>jms_map</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>jms_stream</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>jms_object</td>
</tr>
</tbody>
</table>

The syntax used to express the properties in the variable portion is as follows:

**NameValueLength (MQLONG)**
Length in bytes of the NameValueData string that immediately follows this length field (it does not include its own length). The value set into NameValueLength is always a multiple of 4 (the NameValueData field is padded with space characters to achieve this).

**NameValueData (MQCHARn)**
A single character string, whose length in bytes is given by the preceding NameValueLength field. It contains a folder holding a sequence of properties. Each property is a name/type/value triplet, contained within an XML element whose name is the folder name, as follows:

```xml
<foldername> triplet1 triplet2 ..... tripletn </foldername>
```

The closing `</foldername>` tag can be followed by spaces as padding characters. Each triplet is encoded using an XML-like syntax:

```xml
<name dt='datatype'>value</name>
```

The `dt='datatype'` element is optional and is omitted for many properties, because the datatype is predefined. If it is included, one or more space characters must be included before the `dt=` tag.

- **name** is the name of the property; see Table 19 on page 263.
- **datatype** must match, after folding, one of the literal datatype values in Table 20.
- **value** is a string representation of the value to be conveyed, using the definitions in Table 20.

A null value is encoded using the following syntax:

```xml
<name dt="DataType" xsi:nil="true"/></name>
```

Do not use `xsi:nil="false"`.

Table 20. Property datatype values and definitions

<table>
<thead>
<tr>
<th>Datatype value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>Any sequence of characters excluding &lt; and &amp;</td>
</tr>
<tr>
<td>boolean</td>
<td>The character 0 or 1 (1 = &quot;true&quot;)</td>
</tr>
</tbody>
</table>
### Table 20. Property datatype values and definitions (continued)

<table>
<thead>
<tr>
<th>Datatype value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin.hex</td>
<td>Hexadecimal digits representing octets</td>
</tr>
<tr>
<td>i1</td>
<td>A number, expressed using digits 0..9, with optional sign (no fractions or exponent). Must lie in the range -128 to 127 inclusive</td>
</tr>
<tr>
<td>i2</td>
<td>A number, expressed using digits 0..9, with optional sign (no fractions or exponent). Must lie in the range -32768 to 32767 inclusive</td>
</tr>
<tr>
<td>i4</td>
<td>A number, expressed using digits 0..9, with optional sign (no fractions or exponent). Must lie in the range -2147483648 to 2147483647 inclusive</td>
</tr>
<tr>
<td>i8</td>
<td>A number, expressed using digits 0..9, with optional sign (no fractions or exponent). Must lie in the range -9223372036854775808 to 92233720368547750807 inclusive</td>
</tr>
<tr>
<td>int</td>
<td>A number, expressed using digits 0..9, with optional sign (no fractions or exponent). Must lie in the same range as i8. This can be used in place of one of the i* types if the sender does not want to associate a particular precision with the property</td>
</tr>
<tr>
<td>r4</td>
<td>Floating point number, magnitude (\leq 3.40282347E+38, \geq 1.175E-37) expressed using digits 0..9, optional sign, optional fractional digits, optional exponent</td>
</tr>
<tr>
<td>r8</td>
<td>Floating point number, magnitude (\leq 1.7976931348623E+308, \geq 2.225E-307) expressed using digits 0..9, optional sign, optional fractional digits, optional exponent</td>
</tr>
</tbody>
</table>

A string value can contain spaces. You must use the following escape sequences in a string value:
- `&amp;` for the `&` character
- `&lt;` for the `<` character

You can use the following escape sequences, but they are not required:
- `&gt;` for the `>` character
- `&apos;` for the `'` character
- `&quot;` for the `"` character

### JMS fields and properties with corresponding MQMD fields

Table 21 lists the JMS header fields and Table 22 on page 266 lists the JMS properties that are mapped directly to MQMD fields. Table 23 on page 266 lists the provider specific properties and the MQMD fields that they are mapped to.

#### Table 21: JMS header fields mapping to MQMD fields

<table>
<thead>
<tr>
<th>JMS header field</th>
<th>Java type</th>
<th>MQMD field</th>
<th>C type</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMSDeliveryMode</td>
<td>int</td>
<td>Persistence</td>
<td>MQLONG</td>
</tr>
<tr>
<td>JMSExpiration</td>
<td>long</td>
<td>Expiry</td>
<td>MQLONG</td>
</tr>
<tr>
<td>JMSPriority</td>
<td>int</td>
<td>Priority</td>
<td>MQLONG</td>
</tr>
<tr>
<td>JMSMessageID</td>
<td>String</td>
<td>MessageID</td>
<td>MQBYTE24</td>
</tr>
<tr>
<td>JMSTimestamp</td>
<td>long</td>
<td>PutDate PutTime</td>
<td>MQCHAR8  MQCHAR8</td>
</tr>
<tr>
<td>JMSCorrelationID</td>
<td>String</td>
<td>CorrelId</td>
<td>MQBYTE24</td>
</tr>
</tbody>
</table>
Mapping JMS messages

Table 22. JMS properties mapping to MQMD fields

<table>
<thead>
<tr>
<th>JMS property</th>
<th>Java type</th>
<th>MQMD field</th>
<th>C type</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMSXUserID</td>
<td>String</td>
<td>UserIdentifier</td>
<td>MQCHAR12</td>
</tr>
<tr>
<td>JMSXAppID</td>
<td>String</td>
<td>PutAppIdName</td>
<td>MQCHAR28</td>
</tr>
<tr>
<td>JMSXDeliveryCount</td>
<td>int</td>
<td>BackoutCount</td>
<td>MQLONG</td>
</tr>
<tr>
<td>JMSXGroupID</td>
<td>String</td>
<td>GroupId</td>
<td>MQBYTE24</td>
</tr>
<tr>
<td>JMSXGroupSeq</td>
<td>int</td>
<td>MsgSeqNumber</td>
<td>MQLONG</td>
</tr>
</tbody>
</table>

Table 23. JMS provider specific properties mapping to MQMD fields

<table>
<thead>
<tr>
<th>JMS provider specific property</th>
<th>Java type</th>
<th>MQMD field</th>
<th>C type</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS_IBM_Report_Exception</td>
<td>int</td>
<td>Report</td>
<td>MQLONG</td>
</tr>
<tr>
<td>JMS_IBM_Report_Expiration</td>
<td>int</td>
<td>Report</td>
<td>MQLONG</td>
</tr>
<tr>
<td>JMS_IBM_Report_COA</td>
<td>int</td>
<td>Report</td>
<td>MQLONG</td>
</tr>
<tr>
<td>JMS_IBM_Report_COD</td>
<td>int</td>
<td>Report</td>
<td>MQLONG</td>
</tr>
<tr>
<td>JMS_IBM_Report_PAN</td>
<td>int</td>
<td>Report</td>
<td>MQLONG</td>
</tr>
<tr>
<td>JMS_IBM_Report_NAN</td>
<td>int</td>
<td>Report</td>
<td>MQLONG</td>
</tr>
<tr>
<td>JMS_IBM_Report_Pass_Msg_ID</td>
<td>int</td>
<td>Report</td>
<td>MQLONG</td>
</tr>
<tr>
<td>JMS_IBM_Report_Pass_Correl_ID</td>
<td>int</td>
<td>Report</td>
<td>MQLONG</td>
</tr>
<tr>
<td>JMS_IBM_Report_Discard_Msg</td>
<td>int</td>
<td>Report</td>
<td>MQLONG</td>
</tr>
<tr>
<td>JMS_IBM_MsgType</td>
<td>int</td>
<td>MessageType</td>
<td>MQLONG</td>
</tr>
<tr>
<td>JMS_IBM_Feedback</td>
<td>int</td>
<td>Feedback</td>
<td>MQLONG</td>
</tr>
<tr>
<td>JMS_IBM_Format</td>
<td>String</td>
<td>Format</td>
<td>MQCHAR8</td>
</tr>
<tr>
<td>JMS_IBM_PutAppType</td>
<td>int</td>
<td>PutAppType</td>
<td>MQLONG</td>
</tr>
<tr>
<td>JMS_IBM_Encoding</td>
<td>int</td>
<td>Encoding</td>
<td>MQLONG</td>
</tr>
<tr>
<td>JMS_IBM_Character_Set</td>
<td>String</td>
<td>CodedCharacterSetId</td>
<td>MQLONG</td>
</tr>
<tr>
<td>JMS_IBM_PutDate</td>
<td>String</td>
<td>PutDate</td>
<td>MQCHAR8</td>
</tr>
<tr>
<td>JMS_IBM_PutTime</td>
<td>String</td>
<td>PutTime</td>
<td>MQCHAR8</td>
</tr>
<tr>
<td>JMS_IBM_Last_Msg_In_Group</td>
<td>boolean</td>
<td>MsgFlags</td>
<td>MQLONG</td>
</tr>
</tbody>
</table>

Mapping JMS fields onto WebSphere MQ fields (outgoing messages)

Table 24 on page 267 shows how the JMS header fields are mapped into MQMD/RFH2 fields at send() or publish() time. Table 25 on page 267 shows how JMS properties and Table 26 on page 267 shows how JMS provider specific properties are mapped to MQMD fields at send() or publish() time,

For fields marked Set by Message Object, the value transmitted is the value held in the JMS message immediately before the send() or publish() operation. The value in the JMS message is left unchanged by the operation.

For fields marked Set by Send Method, a value is assigned when the send() or publish() is performed (any value held in the JMS message is ignored). The value in the JMS message is updated to show the value used.
Fields marked as Receive-only are not transmitted and are left unchanged in the message by `send()` or `publish()`.

### Table 24. Outgoing message field mapping

<table>
<thead>
<tr>
<th>JMS header field name</th>
<th>MQMD field used for transmission</th>
<th>Header</th>
<th>Set by</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMSDestination</td>
<td></td>
<td>MQRFH2</td>
<td>Send Method</td>
</tr>
<tr>
<td>JMSDeliveryMode</td>
<td>Persistence</td>
<td>MQRFH2</td>
<td>Send Method</td>
</tr>
<tr>
<td>JMSExpiration</td>
<td>Expiry</td>
<td>MQRFH2</td>
<td>Send Method</td>
</tr>
<tr>
<td>JMSPriority</td>
<td>Priority</td>
<td>MQRFH2</td>
<td>Send Method</td>
</tr>
<tr>
<td>JMSMessageID</td>
<td>MessageID</td>
<td></td>
<td>Send Method</td>
</tr>
<tr>
<td>JMSTimestamp</td>
<td>PutDate/PutTime</td>
<td></td>
<td>Send Method</td>
</tr>
<tr>
<td>JMSReplyCorrelationID</td>
<td>CorrelId</td>
<td>MQRFH2</td>
<td>Message Object</td>
</tr>
<tr>
<td>JMSReplyTo</td>
<td>ReplyToQ/ReplyToQMgr</td>
<td>MQRFH2</td>
<td>Message Object</td>
</tr>
<tr>
<td>JMSRedelivered</td>
<td></td>
<td></td>
<td>Message Object</td>
</tr>
</tbody>
</table>

### Table 25. Outgoing message JMS property mapping

<table>
<thead>
<tr>
<th>JMS property name</th>
<th>MQMD field used for transmission</th>
<th>Header</th>
<th>Set by</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMSXUserID</td>
<td>UserIdentifier</td>
<td></td>
<td>Send Method</td>
</tr>
<tr>
<td>JMSXAppID</td>
<td>PutApplName</td>
<td></td>
<td>Send Method</td>
</tr>
<tr>
<td>JMSDeliveryCount</td>
<td></td>
<td></td>
<td>Receive-only</td>
</tr>
<tr>
<td>JMSXGroupID</td>
<td>GroupId</td>
<td>MQRFH2</td>
<td>Message Object</td>
</tr>
<tr>
<td>JMSXGroupSeq</td>
<td>MsgSeqNumber</td>
<td>MQRFH2</td>
<td>Message Object</td>
</tr>
</tbody>
</table>

### Table 26. Outgoing message JMS provider specific property mapping

<table>
<thead>
<tr>
<th>JMS provider specific property name</th>
<th>MQMD field used for transmission</th>
<th>Header</th>
<th>Set by</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS_IBM_Report_Exception</td>
<td>Report</td>
<td></td>
<td>Message Object</td>
</tr>
<tr>
<td>JMS_IBM_Report_Expiration</td>
<td>Report</td>
<td></td>
<td>Message Object</td>
</tr>
<tr>
<td>JMS_IBM_Report_COA/COD</td>
<td>Report</td>
<td></td>
<td>Message Object</td>
</tr>
<tr>
<td>JMS_IBM_Report_NAN/PAN</td>
<td>Report</td>
<td></td>
<td>Message Object</td>
</tr>
<tr>
<td>JMS_IBM_Report_Pass_Correl_ID</td>
<td>Report</td>
<td></td>
<td>Message Object</td>
</tr>
<tr>
<td>JMS_IBM_Report_Discard_Msg</td>
<td>Report</td>
<td></td>
<td>Message Object</td>
</tr>
<tr>
<td>JMS_IBM_MsgType</td>
<td>MsgType</td>
<td></td>
<td>Message Object</td>
</tr>
<tr>
<td>JMS_IBM_Feedback</td>
<td>Feedback</td>
<td></td>
<td>Message Object</td>
</tr>
<tr>
<td>JMS_IBM_Format</td>
<td>Format</td>
<td></td>
<td>Message Object</td>
</tr>
<tr>
<td>JMS_IBM_PutApplType</td>
<td>PutApplType</td>
<td></td>
<td>Send Method</td>
</tr>
<tr>
<td>JMS_IBM_Encoding</td>
<td>Encoding</td>
<td></td>
<td>Message Object</td>
</tr>
<tr>
<td>JMS_IBM_Character_Set</td>
<td>CodedCharacterSetId</td>
<td></td>
<td>Message Object</td>
</tr>
<tr>
<td>JMS_IBM_PutDate</td>
<td>PutDate</td>
<td></td>
<td>Send Method</td>
</tr>
<tr>
<td>JMS_IBM_PutTime</td>
<td>PutTime</td>
<td></td>
<td>Send Method</td>
</tr>
</tbody>
</table>
Mapping JMS messages

Table 26. Outgoing message JMS provider specific property mapping (continued)

<table>
<thead>
<tr>
<th>JMS provider specific property name</th>
<th>MQMD field used for transmission</th>
<th>Header</th>
<th>Set by</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS_IBM_Last_Msg_In_Group</td>
<td>MsgFlags</td>
<td></td>
<td>Message Object</td>
</tr>
</tbody>
</table>

Mapping JMS header fields at send() or publish()

The following notes relate to the mapping of JMS fields at send() or publish():

**JMSDestination to MQRFH2**

This is stored as a string that serializes the salient characteristics of the destination object, so that a receiving JMS can reconstitute an equivalent destination object. The MQRFH2 field is encoded as URI (see “uniform resource identifiers” on page 204 for details of the URI notation).

**JMSReplyTo to MQMD ReplyToQ, ReplyToQMgr, MQRFH2**

The queue and queue manager name are copied to the MQMD ReplyToQ and ReplyToQMgr fields respectively. The destination extension information (other useful details that are kept in the destination object) is copied into the MQRFH2 field. The MQRFH2 field is encoded as a URI (see “uniform resource identifiers” on page 204 for details of the URI notation).

**JMSDeliveryMode to MQMD Persistence**

The JMSDeliveryMode value is set by the send() or publish() Method or MessageProducer, unless the Destination Object overrides it. The JMSDeliveryMode value is mapped to the MQMD Persistence field as follows:

- JMS value PERSISTENT is equivalent to MQPER_PERSISTENT
- JMS value NON_PERSISTENT is equivalent to MQPER_NOT_PERSISTENT

If the MQQueue persistence property is not set to JMSC.MQJMS_PER_QDEF, the delivery mode value is also encoded in the MQRFH2.

**JMSExpiration to/from MQMD Expiry, MQRFH2**

JMSExpiration stores the time to expire (the sum of the current time and the time to live), whereas MQMD stores the time to live. Also, JMSExpiration is in milliseconds, but MQMD.expiry is in centiseconds.

- If the send() method sets an unlimited time to live, MQMD Expiry is set to MQEI_UNLIMITED, and no JMSExpiration is encoded in the MQRFH2.
- If the send() method sets a time to live that is less than 214748364.7 seconds (about 7 years), the time to live is stored in MQMD. Expiry, and the expiration time (in milliseconds), are encoded as an i8 value in the MQRFH2.
- If the send() method sets a time to live greater than 214748364.7 seconds, MQMD.Expiry is set to MQEI_UNLIMITED. The true expiration time in milliseconds is encoded as an i8 value in the MQRFH2.

**JMSPriority to MQMD Priority**

Directly map JMSPriority value (0-9) onto MQMD priority value (0-9). If JMSPriority is set to a non-default value, the priority level is also encoded in the MQRFH2.

**JMSMessageID from MQMD MessageID**

All messages sent from JMS have unique message identifiers assigned by
WebSphere MQ. The value assigned is returned in the MQMD messageId field after the MQPUT call, and is passed back to the application in the JMSMessageID field. The WebSphere MQ messageId is a 24-byte binary value, whereas the JMSMessageID is a string. The JMSMessageID is composed of the binary messageId value converted to a sequence of 48 hexadecimal characters, prefixed with the characters ID.: JMS provides a hint that can be set to disable the production of message identifiers. This hint is ignored, and a unique identifier is assigned in all cases. Any value that is set into the JMSMessageID field before a send() is overwritten.

**JMSTimestamp to MQRFH2**
During a send, the JMSTimestamp field is set according to the JVM’s clock. This value is set into the MQRFH2. Any value that is set into the JMSTimestamp field before a send() is overwritten. See also the JMS IBM PutDate and JMS IBM PutTime properties.

**JMSType to MQRFH2**
This string is set into the MQRFH2 mcd.Type field. If it is in URI format, it can also affect mcd.Set and mcd.Fmt fields. See also “Connecting to other products,” on page 469.

**JMSCorrelationID to MQMD CorrelId, MQRFH2**
The JMSCorrelationID can hold one of the following:

A provider specific message ID
This is a message identifier from a message previously sent or received, and so should be a string of 48 hexadecimal digits that are prefixed with ID:. The prefix is removed, the remaining characters are converted into binary, and then they are set into the MQMD CorrelId field. No CorrelId value is encoded in the MQRFH2.

A provider-native byte[] value
The value is copied into the MQ MD CorrelId field - padded with nulls, or truncated to 24 bytes if necessary. No CorrelId value is encoded in the MQRFH2.

An application-specific string
The value is copied into the MQRFH2. The first 24 bytes of the string, in UTF8 format, are written into the MQMD CorrelID.

**Mapping JMS property fields**
These notes refer to the mapping of JMS property fields in WebSphere MQ messages:

**JMSXUserID from MQMD UserIdentifier**
JMSXUserID is set on return from send call.

**JMSXAppID from MQMD PutApplName**
JMSXAppID is set on return from send call.

**JMSXGroupID to MQRFH2 (point-to-point)**
For point-to-point messages, the JMSXGroupID is copied into the MQMD GroupID field. If the JMSXGroupID starts with the prefix ID:, it is converted into binary. Otherwise, it is encoded as a UTF8 string. The value is padded or truncated if necessary to a length of 24 bytes. The MQMF_MSG_IN_GROUP flag is set.

**JMSXGroupID to MQRFH2 (publish/subscribe)**
For publish/subscribe messages, the JMSXGroupID is copied into the MQRFH2 as a string.
Mapping JMS messages

JMSXGroupSeq MQMD MsgSeqNumber (point-to-point)
For point-to-point messages, the JMSXGroupSeq is copied into the MQMD MsgSeqNumber field. The MQMF_MSG_IN_GROUP flag is set.

JMSXGroupSeq MQMD MsgSeqNumber (publish/subscribe)
For publish/subscribe messages, the JMSXGroupSeq is copied into the MQRFH2 as an i4.

Mapping JMS provider-specific fields
The following notes refer to the mapping of JMS Provider specific fields into WebSphere MQ messages:

JMS_IBM_Report_<name> to MQMD Report
A JMS application can set the MQMD Report options, using the following JMS_IBM_Report_XXX properties. The single MQMD is mapped to several JMS_IBM_Report_XXX properties. The application must set the value of these properties to the standard WebSphere MQ MQRO_constants (included in com.ibm.mq.MQC). So, for example, to request COD with full Data, the application must set JMS_IBM_Report_COD to the value MQC.MQRO_COD_WITH_FULL_DATA.

JMS_IBM_Report_Exception
MQRO_EXCEPTION or
MQRO_EXCEPTION_WITH_DATA or
MQRO_EXCEPTION_WITH_FULL_DATA

JMS_IBM_Report_Expiration
MQRO_EXPIRATION or
MQRO_EXPIRATION_WITH_DATA or
MQRO_EXPIRATION_WITH_FULL_DATA

JMS_IBM_Report_COA
MQRO_COA or
MQRO_COA_WITH_DATA or
MQRO_COA_WITH_FULL_DATA

JMS_IBM_Report_COD
MQRO_COD or
MQRO_COD_WITH_DATA or
MQRO_COD_WITH_FULL_DATA

JMS_IBM_Report_PAN
MQRO_PAN

JMS_IBM_Report_NAN
MQRO_NAN

JMS_IBM_Report_Pass_Msg_ID
MQRO_PASS_MSG_ID

JMS_IBM_Report_Pass_Correl_ID
MQRO_PASS_CORREL_ID

JMS_IBM_Report_Discard_Msg
MQRO_DISCARD_MSG

JMS_IBM_MsgType to MQMD MsgType
Value maps directly onto MQMD MsgType. If the application has not set
an explicit value of JMS.ibm.MsgType, a default value is used. This default value is determined as follows:

- If JMSReplyTo is set to a WebSphere MQ queue destination, MSGType is set to the value MGMT_REQUEST
- If JMSReplyTo is not set, or is set to anything other than a WebSphere MQ queue destination, MsgType is set to the value MGMT_DATAGRAM

JMS.ibm.Feedback to MQMD Feedback
Value maps directly onto MQMD Feedback.

JMS.ibm.Format to MQMD Format
Value maps directly onto MQMD Format.

JMS.ibm.Encoding to MQMD Encoding
If set, this property overrides the numeric encoding of the Destination Queue or Topic.

JMS.ibm.Character_Set to MQMD CodedCharacterSetId
If set, this property overrides the coded character set property of the Destination Queue or Topic.

JMS.ibm.PutDate from MQMD PutDate
The value of this property is set, during send, directly from the PutDate field in the MQMD. Any value that is set into the JMS.ibm.PutDate property before a send is overwritten. This field is a String of eight characters, in the WebSphere MQ Date format of YYYYMMDD. This property can be used in conjunction with the JMS.ibm.PutTime property to determine the time the message was put according to the queue manager.

JMS.ibm.PutTime from MQMD PutTime
The value of this property is set, during send, directly from the PutTime field in the MQMD. Any value that is set into the JMS.ibm.PutTime property before a send is overwritten. This field is a String of eight characters, in the WebSphere MQ Time format of HHMMSTH. This property can be used in conjunction with the JMS.ibm.PutDate property to determine the time the message was put according to the queue manager.

JMS.ibm.Last_Msg_In_Group to MQMD MsgFlags
For point-to-point messaging, this boolean value maps to the MQMF_LAST_MSG_IN_GROUP flag in the MQMD MsgFlags field. It is normally used in conjunction with the JMSXGroupID and JMSXGroupSeq properties to indicate to a legacy WebSphere MQ application that this is the last message in a group. This property is ignored for publish/subscribe messaging.

### Mapping WebSphere MQ fields onto JMS fields (incoming messages)

Table 27 on page 272 shows how JMS header fields and Table 28 on page 272 shows how JMS property fields are mapped into MQMD/MQRFH2 fields at send() or publish() time. Table 29 on page 272 shows how JMS provider specific properties are mapped.
### Table 27. Incoming message JMS header field mapping

<table>
<thead>
<tr>
<th>JMS header field name</th>
<th>MQMD field retrieved from</th>
<th>MQRFH2 field retrieved from</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMSDestination</td>
<td></td>
<td>jms.Dst</td>
</tr>
<tr>
<td>JMSDeliveryMode</td>
<td>Persistence</td>
<td>jms.Dlv</td>
</tr>
<tr>
<td>JMSExpiration</td>
<td></td>
<td>jms.Exp</td>
</tr>
<tr>
<td>JMSPriority</td>
<td>Priority</td>
<td></td>
</tr>
<tr>
<td>JMSMessageID</td>
<td>MessageID</td>
<td></td>
</tr>
<tr>
<td>JMSTimestamp</td>
<td>PutDate</td>
<td>jms.Tms</td>
</tr>
<tr>
<td></td>
<td>PutTime</td>
<td></td>
</tr>
<tr>
<td>JMSCorrelationID</td>
<td>CorrelId</td>
<td>jms.Cid</td>
</tr>
<tr>
<td>JMSReplyTo</td>
<td>ReplyTo</td>
<td>jms.Rto</td>
</tr>
<tr>
<td></td>
<td>ReplyToQMg</td>
<td></td>
</tr>
<tr>
<td>JMSType</td>
<td></td>
<td>mcd.Type, mcd.Set, mcdFmt</td>
</tr>
<tr>
<td>JMSRedelivered</td>
<td>BackoutCount</td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
1. For properties that can have values retrieved from the MQRFH2 or the MQMD, if both are available, the setting in the MQRFH2 is used.

### Table 28. Incoming message property mapping

<table>
<thead>
<tr>
<th>JMS property name</th>
<th>MQMD field retrieved from</th>
<th>MQRFH2 field retrieved from</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMSXUserID</td>
<td>UserIdentifier</td>
<td></td>
</tr>
<tr>
<td>JMSXAppID</td>
<td>PutAppName</td>
<td></td>
</tr>
<tr>
<td>JMSXDeliveryCount</td>
<td>BackoutCount</td>
<td></td>
</tr>
<tr>
<td>JMSXGroupID</td>
<td>GroupId</td>
<td>jms.Gid</td>
</tr>
<tr>
<td>JMSXGroupSeq</td>
<td>MsgSeqNumber</td>
<td>jms.Seq</td>
</tr>
</tbody>
</table>

**Notes:**
1. For properties that can have values retrieved from the MQRFH2 or the MQMD, if both are available, the setting in the MQRFH2 is used.

### Table 29. Incoming message provider specific JMS property mapping

<table>
<thead>
<tr>
<th>JMS property name</th>
<th>MQMD field retrieved from</th>
<th>MQRFH2 field retrieved from</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS_FILE_IBM_Report_Exception</td>
<td>Report</td>
<td></td>
</tr>
<tr>
<td>JMS_FILE_IBM_Report_Expiration</td>
<td>Report</td>
<td></td>
</tr>
<tr>
<td>JMS_FILE_IBM_Report_COA</td>
<td>Report</td>
<td></td>
</tr>
<tr>
<td>JMS_FILE_IBM_Report_COD</td>
<td>Report</td>
<td></td>
</tr>
<tr>
<td>JMS_FILE_IBM_Report_PAN</td>
<td>Report</td>
<td></td>
</tr>
<tr>
<td>JMS_FILE_IBM_Report_NAN</td>
<td>Report</td>
<td></td>
</tr>
<tr>
<td>JMS_FILE_IBM_Report_Pass_Msg_ID</td>
<td>Report</td>
<td></td>
</tr>
<tr>
<td>JMS_FILE_IBM_Report_Pass_Correl_ID</td>
<td>Report</td>
<td></td>
</tr>
<tr>
<td>JMS_FILE_IBM_Report_Discard_Msg</td>
<td>Report</td>
<td></td>
</tr>
</tbody>
</table>
Table 29. Incoming message provider specific JMS property mapping (continued)

<table>
<thead>
<tr>
<th>JMS property name</th>
<th>MQMD field retrieved from</th>
<th>MQRFH2 field retrieved from</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS_IW_MsgType</td>
<td>MsgType</td>
<td></td>
</tr>
<tr>
<td>JMS_IW_Feedback</td>
<td>Feedback</td>
<td></td>
</tr>
<tr>
<td>JMS_IW_Format</td>
<td>Format</td>
<td></td>
</tr>
<tr>
<td>JMS_IW_PutApplType</td>
<td>PutApplType</td>
<td></td>
</tr>
<tr>
<td>JMS_IW_Encoding</td>
<td>Encoding</td>
<td></td>
</tr>
<tr>
<td>JMS_IW_Character_Set</td>
<td>CodedCharacterSetId</td>
<td></td>
</tr>
<tr>
<td>JMS_IW_PutDate</td>
<td>PutDate</td>
<td></td>
</tr>
<tr>
<td>JMS_IW_PutTime</td>
<td>PutTime</td>
<td></td>
</tr>
<tr>
<td>JMS_IW_Last_Msg_In_Group</td>
<td>MsgFlags</td>
<td></td>
</tr>
</tbody>
</table>

1. Only set if the incoming message is a Bytes Message.

Mapping JMS to a native WebSphere MQ application

This section describes what happens if you send a message from a JMS client application to a traditional WebSphere MQ application with no knowledge of MQRFH2 headers. Figure 5 shows the mapping.

The administrator indicates that the JMS client is communicating with such an application by setting the WebSphere MQ destination’s TargetClient value to JMSC MQJMS_CLIENT_NONJMS_MQ. This indicates that no MQRFH2 field is to be produced. Note that if this is not done, the receiving application must be able to handle the MQRFH2 field.

The mapping from JMS to MQMD targeted at a native WebSphere MQ application is the same as mapping from JMS to MQMD targeted at a true JMS client. If JMS receives a WebSphere MQ message with the MQMD format field set to other than MQFMT_RFH2, data is being received from a non-JMS application. If the format is MQFMT_STRING, the message is received as a JMS text message. Otherwise, it is received as a JMS bytes message. Because there is no MQRFH2, only those JMS properties that are transmitted in the MQMD can be restored.

Figure 5. How JMS messages are transformed to WebSphere MQ messages (no MQRFH2 header)

Message body

This section discusses the encoding of the message body itself. The encoding depends on the type of JMS message:
Mapping JMS messages

**ObjectMessage**

is an object serialized by the Java Runtime in the normal way.

**TextMessage**

is an encoded string. For an outgoing message, the string is encoded in the character set given by the destination object. This defaults to UTF8 encoding (the UTF8 encoding starts with the first character of the message; there is no length field at the start). It is, however, possible to specify any other character set supported by WebSphere MQ Java. Such character sets are used mainly when you send a message to a non-JMS application.

If the character set is a double-byte set (including UTF16), the destination object’s integer encoding specification determines the order of the bytes.

An incoming message is interpreted using the character set and encoding that are specified in the message itself. These specifications are in the last WebSphere MQ header (or MQMD if there are no headers). For JMS messages, the last header is usually the MQRFH2.

**BytesMessage**

is, by default, a sequence of bytes as defined by the JMS 1.0.2 specification and associated Java documentation.

For an outgoing message that was assembled by the application itself, the destination object’s encoding property can be used to override the encodings of integer and floating point fields contained in the message. For example, you can request that floating point values are stored in S/390 rather than IEEE format.

An incoming message is interpreted using the numeric encoding specified in the message itself. This specification is in the rightmost WebSphere MQ header (or MQMD if there are no headers). For JMS messages, the rightmost header is usually the MQRFH2.

If a BytesMessage is received, and is re-sent without modification, its body is transmitted byte for byte, as it was received. The destination object’s encoding property has no effect on the body. The only string-like entity that can be sent explicitly in a BytesMessage is a UTF8 string. This is encoded in Java UTF8 format, and starts with a 2-byte length field. The destination object’s character set property has no effect on the encoding of an outgoing BytesMessage. The character set value in an incoming WebSphere MQ message has no effect on the interpretation of that message as a JMS BytesMessage.

Non-Java applications are unlikely to recognize the Java UTF8 encoding. Therefore, for a JMS application to send a BytesMessage that contains text data, the application itself must convert its strings to byte arrays, and write these byte arrays into the BytesMessage.

**MapMessage**

is a string containing a set of XML name/type/value triplets, encoded as:

```
<map><elementName1 dt='datatype'>value</elementName1>
<elementName2 dt='datatype'>value</elementName2>.....
</map>
```

where datatype can take one of the values described in Table 20 on page 264 and string is the default datatype, so dt='string' is omitted.

The character set used to encode or interpret the XML string that makes up the MapMessage body is determined following the rules that apply to a TextMessage.
### StreamMessage

is like a map, but without element names:

```xml
<stream><elt dt='datatype'>value</elt>
<elt dt='datatype'>value</elt>.....</stream>
```

Every element is sent using the same tag name (elt). The default type is string, so `dt='string'` is omitted for string elements.

The character set used to encode or interpret the XML string that makes up the StreamMessage body is determined following the rules that apply to a TextMessage.

The MQRFH2.format field is set as follows:

- **MQFMT_NONE**
  - for ObjectMessage, BytesMessage, or messages with no body.

- **MQFMT_STRING**
  - for TextMessage, StreamMessage, or MapMessage.
Mapping JMS messages
Chapter 14. WebSphere MQ JMS Application Server Facilities

WebSphere MQ JMS supports the Application Server Facilities (ASF) that are specified in the Java Message Service 1.0.2 specification (see Sun’s Java Web site at http://java.sun.com). This specification identifies three roles within this programming model:

- The JMS provider supplies ConnectionConsumer and advanced Session functionality.
- The application server supplies ServerSessionPool and ServerSession functionality.
- The client application uses the functionality that the JMS provider and application server supply.

This chapter does not apply if you use a direct connection to WebSphere MQ Event Broker.

The following sections contain details about how WebSphere MQ JMS implements ASF:

- "ASF classes and functions" describes how WebSphere MQ JMS implements the ConnectionConsumer class and advanced functionality in the Session class.
- "Application server sample code” on page 283 describes the sample ServerSessionPool and ServerSession code that is supplied with WebSphere MQ JMS.
- "Examples of ASF use” on page 287 describes supplied ASF samples and examples of ASF use from the perspective of a client application.

Note: The Java Message Service 1.0.2 specification for ASF also describes JMS support for distributed transactions using the X/Open XA protocol. For details of the XA support that WebSphere MQ JMS provides, see Appendix E, “JMS JTA/XA interface with WebSphere Application Server V4,” on page 475.

ASF classes and functions

WebSphere MQ JMS implements the ConnectionConsumer class and advanced functionality in the Session class. For details, see:

- "ConnectionConsumer” on page 318
- "QueueConnection” on page 379
- “Session” on page 393
- “TopicConnection” on page 420

ConnectionConsumer

The JMS specification enables an application server to integrate closely with a JMS implementation by using the ConnectionConsumer interface. This feature provides concurrent processing of messages. Typically, an application server creates a pool of threads, and the JMS implementation makes messages available to these threads. A JMS-aware application server can use this feature to provide high-level messaging functionality, such as message processing beans.
Normal applications do not use the ConnectionConsumer, but expert JMS clients might use it. For such clients, the ConnectionConsumer provides a high-performance method to deliver messages concurrently to a pool of threads. When a message arrives on a queue or a topic, JMS selects a thread from the pool and delivers a batch of messages to it. To do this, JMS runs an associated MessageListener’s `onMessage()` method.

You can achieve the same effect by constructing multiple Session and MessageConsumer objects, each with a registered MessageListener. However, the ConnectionConsumer provides better performance, less use of resources, and greater flexibility. In particular, fewer Session objects are required.

To help you develop applications that use ConnectionConsumers, WebSphere MQ JMS provides a fully-functioning example implementation of a pool. You can use this implementation without any changes, or adapt it to suit the specific needs of the application.

### Planning an application

This section tells you how to plan an application including:

- “General principles for point-to-point messaging”
- “General principles for publish/subscribe messaging” on page 279
- “Handling poison messages” on page 280
- “Removing messages from the queue” on page 281

### General principles for point-to-point messaging

When an application creates a ConnectionConsumer from a QueueConnection object, it specifies a JMS queue object and a selector string. The ConnectionConsumer then begins to provide messages to sessions in the associated ServerSessionPool. Messages arrive on the queue, and if they match the selector, they are delivered to sessions in the associated ServerSessionPool.

In WebSphere MQ terms, the queue object refers to either a QLOCAL or a QALIAS on the local queue manager. If it is a QALIAS, that QALIAS must refer to a QLOCAL. The fully-resolved WebSphere MQ QLOCAL is known as the underlying QLOCAL. A ConnectionConsumer is said to be active if it is not closed and its parent QueueConnection is started.

It is possible for multiple ConnectionConsumers, each with different selectors, to run against the same underlying QLOCAL. To maintain performance, unwanted messages must not accumulate on the queue. Unwanted messages are those for which no active ConnectionConsumer has a matching selector. You can set the QueueConnectionFactory so that these unwanted messages are removed from the queue (for details, see “Removing messages from the queue” on page 281). You can set this behavior in one of two ways:

- Use the JMS administration tool to set the QueueConnectionFactory to MRET(NO).
- In your program, use:
  ```java
  MQQueueConnectionFactory.setMessageRetention(JMSC.MQJMS_MRET_NO)
  ```

If you do not change this setting, the default is to retain such unwanted messages on the queue.

It is possible that ConnectionConsumers that target the same underlying QLOCAL could be created from multiple QueueConnection objects. However, for
performance reasons, we recommend that multiple JVMs do not create ConnectionConsumers against the same underlying QLOCAL.

When you set up the WebSphere MQ queue manager, consider the following points:

- The underlying QLOCAL must be enabled for shared input. To do this, use the following MQSC command:
  
  `ALTER QLOCAL(your.qlocal.name) SHARE GET(ENABLED)`

- Your queue manager must have an enabled dead-letter queue. If a ConnectionConsumer experiences a problem when it puts a message on the dead-letter queue, message delivery from the underlying QLOCAL stops. To define a dead-letter queue, use:
  
  `ALTER QMGR DEADQ(your.dead.letter.queue.name)`

- The user that runs the ConnectionConsumer must have authority to perform MQOPEN with MQOO_SAVE_ALL_CONTEXT and MQOO_PASS_ALL_CONTEXT. For details, see the WebSphere MQ documentation for your specific platform.

- If unwanted messages are left on the queue, they degrade the system performance. Therefore, plan your message selectors so that between them, the ConnectionConsumers will remove all messages from the queue.

For details about MQSC commands, see the [WebSphere MQ Script (MQSC) Command Reference](#).

### General principles for publish/subscribe messaging

When an application creates a ConnectionConsumer from a TopicConnectionFactory object, it specifies a Topic object and a selector string. The ConnectionConsumer then begins to receive messages that match the selector on that Topic.

Alternatively, an application can create a durable ConnectionConsumer that is associated with a specific name. This ConnectionConsumer receives messages that have been published on the Topic since the durable ConnectionConsumer was last active. It receives all such messages that match the selector on the Topic.

For non-durable subscriptions, a separate queue is used for ConnectionConsumer subscriptions. The CCSUB configurable option on the TopicConnectionFactory specifies the queue to use. Normally, the CCSUB specifies a single queue for use by all ConnectionConsumers that use the same TopicConnectionFactory. However, it is possible to make each ConnectionConsumer generate a temporary queue by specifying a queue name prefix followed by a `*`.

For durable subscriptions, the CCDSUB property of the Topic specifies the queue to use. Again, this can be a queue that already exists or a queue name prefix followed by a `*`. If you specify a queue that already exists, all durable ConnectionConsumers that subscribe to the Topic use this queue. If you specify a queue name prefix followed by a `*`, a queue is generated the first time that a durable ConnectionConsumer is created with a given name. This queue is reused later when a durable ConnectionConsumer is created with the same name.

When you set up the WebSphere MQ queue manager, consider the following points:

- Your queue manager must have an enabled dead-letter queue. If a ConnectionConsumer experiences a problem when it puts a message on the dead-letter queue, message delivery from the underlying QLOCAL stops. To define a dead-letter queue, use:
ASF classes and functions

ALTER QMGR DEADQ(your.dead.letter.queue.name)

- The user that runs the ConnectionConsumer must have authority to perform MQOPEN with MQOO_SAVE_ALL_CONTEXT and MQOO_PASS_ALL_CONTEXT. For details, see the WebSphere MQ documentation for your platform.

- You can optimize performance for an individual ConnectionConsumer by creating a separate, dedicated, queue for it. This is at the cost of extra resource usage.

Handling poison messages

Sometimes, a badly-formatted message arrives on a queue. Such a message might make the receiving application fail and back out the receipt of the message. In this situation, such a message might be received, then returned to the queue, repeatedly. These messages are known as poison messages. The ConnectionConsumer must be able to detect poison messages and reroute them to an alternative destination.

When an application uses ConnectionConsumers, the circumstances in which a message is backed out depend on the session that the application server provides:

- When the session is non-transacted, with AUTO_ACKNOWLEDGE or DUPS_OK_ACKNOWLEDGE, a message is backed out only after a system error, or if the application terminates unexpectedly.

- When the session is non-transacted with CLIENT_ACKNOWLEDGE, unacknowledged messages can be backed out by the application server calling Session.recover(). Typically, the client implementation of MessageListener or the application server calls Message.acknowledge(). Message.acknowledge() acknowledges all messages delivered on the session so far.

- When the session is transacted, the application server usually commits the session. If the application server detects an error, it may choose to back out one or more messages.

- If the application server supplies an XASession, messages are committed or backed out depending on a distributed transaction. The application server takes responsibility for completing the transaction.

The WebSphere MQ queue manager keeps a record of the number of times that each message has been backed out. When this number reaches a configurable threshold, the ConnectionConsumer requeues the message on a named backout queue. If this requeue fails for any reason, the message is removed from the queue and either requeued to the dead-letter queue, or discarded. See "Removing messages from the queue" on page 281 for more details.

On most platforms, the threshold and requeue queue are properties of the WebSphere MQ QLOCAL. For point-to-point messaging, this is the underlying QLOCAL. For publish/subscribe messaging, this is the CCSUB queue defined on the TopicConnectionFactory, or the CCDSUB queue defined on the Topic. To set the threshold and requeue queue properties, issue the following MQSC command:

ALTER QLOCAL(your.queue.name) BOTHRESH(threshold) BOQUEUE(your.requeue.queue.name)

For publish/subscribe messaging, if your system creates a dynamic queue for each subscription, these settings are obtained from the WebSphere MQ JMS model queue. To alter these settings, you can use:

ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(threshold) BOQUEUE(your.requeue.queue.name)
If the threshold is zero, poison message handling is disabled, and poison messages remain on the input queue. Otherwise, when the backout count reaches the threshold, the message is sent to the named requeue queue. If the backout count reaches the threshold, but the message cannot go to the requeue queue, the message is sent to the dead-letter queue or discarded. This situation occurs if the requeue queue is not defined, or if the ConnectionConsumer cannot send the message to the requeue queue. On some platforms, you cannot specify the threshold and requeue queue properties. On these platforms, messages are sent to the dead-letter queue, or discarded, when the backout count reaches 20. See “Removing messages from the queue” for further details.

Removing messages from the queue
When an application uses ConnectionConsumers, JMS might need to remove messages from the queue in a number of situations:

Badly formatted message
A message might arrive that JMS cannot parse.

Poison message
A message might reach the backout threshold, but the ConnectionConsumer fails to requeue it on the backout queue.

No interested ConnectionConsumer
For point-to-point messaging, when the QueueConnectionFactory is set so that it does not retain unwanted messages, a message arrives that is unwanted by any of the ConnectionConsumers.

In these situations, the ConnectionConsumer attempts to remove the message from the queue. The disposition options in the report field of the message’s MQMD set the exact behavior. These options are:

MQRO_DEAD_LETTER_Q
The message is requeued to the queue manager’s dead-letter queue. This is the default.

MQRO_DISCARD_MSG
The message is discarded.

The ConnectionConsumer also generates a report message, and this also depends on the report field of the message’s MQMD. This message is sent to the message’s ReplyToQ on the ReplyToQmgr. If there is an error while the report message is being sent, the message is sent to the dead-letter queue instead. The exception report options in the report field of the message’s MQMD set details of the report message. These options are:

MQRO_EXCEPTION
A report message is generated that contains the MQMD of the original message. It does not contain any message body data.

MQRO_EXCEPTION_WITH_DATA
A report message is generated that contains the MQMD, any MQ headers, and 100 bytes of body data.

MQRO_EXCEPTION_WITH_FULL_DATA
A report message is generated that contains all data from the original message.

default
No report message is generated.

When report messages are generated, the following options are honored:
ASF classes and functions

- MQRO_NEW_MSG_ID
- MQRO_PASS_MSG_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_PASS_CORREL_ID

If a ConnectionConsumer cannot follow the disposition options or exception report options in the message’s MQMD, its action depends on the persistence of the message. If the message is non-persistent, the message is discarded and no report message is generated. If the message is persistent, delivery of all messages from the QLOCAL stops.

It is important to define a dead-letter queue, and to check it regularly to ensure that no problems occur. Particularly, ensure that the dead-letter queue does not reach its maximum depth, and that its maximum message size is large enough for all messages.

When a message is requeued to the dead-letter queue, it is preceded by a WebSphere MQ dead-letter header (MQDLH). See the WebSphere MQ Application Programming Reference for details about the format of the MQDLH. You can identify messages that a ConnectionConsumer has placed on the dead-letter queue, or report messages that a ConnectionConsumer has generated, by the following fields:

- PutApplType is MQAT_JAVA (0x1C)
- PutApplName is “MQ JMS ConnectionConsumer”

These fields are in the MQDLH of messages on the dead-letter queue, and the MQMD of report messages. The feedback field of the MQMD, and the Reason field of the MQDLH, contain a code describing the error. For details about these codes, see “Error handling.” Other fields are as described in the WebSphere MQ Application Programming Reference.

Error handling

This section covers various aspects of error handling, including “Recovering from error conditions” and “Reason and feedback codes” on page 283.

Recovering from error conditions

If a ConnectionConsumer experiences a serious error, message delivery to all ConnectionConsumers with an interest in the same QLOCAL stops. Typically, this occurs if the ConnectionConsumer cannot requeue a message to the dead-letter queue, or it experiences an error when reading messages from the QLOCAL.

When this occurs, any ExceptionListener that is registered with the affected Connection is notified.

You can use these to identify the cause of the problem. In some cases, the system administrator must intervene to resolve the problem.

There are two ways in which an application can recover from these error conditions:

- Call close() on all affected ConnectionConsumers. The application can create new ConnectionConsumers only after all affected ConnectionConsumers are closed and any system problems are resolved.
- Call stop() on all affected Connections. Once all Connections are stopped and any system problems are resolved, the application should be able to start() all Connections successfully.
Reason and feedback codes
To determine the cause of an error, you can use:
- The feedback code in any report messages
- The reason code in the MQDLH of any messages in the dead-letter queue

ConnectionConsumers generate the following reason codes.

**MQRC_BACKOUT_THRESHOLD_REACHED (0x93A; 2362)**

<table>
<thead>
<tr>
<th>Cause</th>
<th>The message has reached the Backout Threshold defined on the QLOCAL, but no Backout Queue is defined. On platforms where you cannot define the Backout Queue, the message has reached the JMS-defined backout threshold of 20.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>If this is not wanted, define the Backout Queue for the relevant QLOCAL. Also look for the cause of the multiple backouts.</td>
</tr>
</tbody>
</table>

**MQRC_MSG_NOT_MATCHED (0x93B; 2363)**

<table>
<thead>
<tr>
<th>Cause</th>
<th>In point-to-point messaging, there is a message that does not match any of the selectors for the ConnectionConsumers monitoring the queue. To maintain performance, the message is requeued to the dead-letter queue.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>To avoid this situation, ensure that ConnectionConsumers using the queue provide a set of selectors that deal with all messages, or set the QueueConnectionFactory to retain messages. Alternatively, investigate the source of the message.</td>
</tr>
</tbody>
</table>

**MQRC_JMS_FORMAT_ERROR (0x93C; 2364)**

<table>
<thead>
<tr>
<th>Cause</th>
<th>JMS cannot interpret the message on the queue.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Investigate the origin of the message. JMS usually delivers messages of an unexpected format as a BytesMessage or TextMessage. Occasionally, this fails if the message is very badly formatted.</td>
</tr>
</tbody>
</table>

Other codes that appear in these fields are caused by a failed attempt to requeue the message to a Backout Queue. In this situation, the code describes the reason that the requeue failed. To diagnose the cause of these errors, refer to the [WebSphere MQ Application Programming Reference](#).

If the report message cannot be put on the ReplyToQ, it is put on the dead-letter queue. In this situation, the feedback field of the MQMD is filled in as described above. The reason field in the MQDLH explains why the report message could not be placed on the ReplyToQ.

**Application server sample code**

[Figure 6 on page 284](#) summarizes the principles of ServerSessionPool and ServerSession functionality.
1. The ConnectionConsumers get message references from the queue.
2. Each ConnectionConsumer selects specific message references.
3. The ConnectionConsumer buffer holds the selected message references.
4. The ConnectionConsumer requests one or more ServerSessions from the ServerSessionPool.
5. ServerSessions are allocated from the ServerSessionPool.
6. The ConnectionConsumer assigns message references to the ServerSessions and starts the ServerSession threads running.
7. Each ServerSession retrieves its referenced messages from the queue. It passes them to the onMessage method from the MessageListener that is associated with the JMS Session.
8. After it completes its processing, the ServerSession is returned to the pool.
Application server sample code

Normally, the application server supplies ServerSessionPool and ServerSession functionality. However, WebSphere MQ JMS is supplied with a simple implementation of these interfaces, with program source. These samples are in the following directory, where <install_dir> is the installation directory for WebSphere MQ JMS:

<install_dir>/samples/jms/asf

These samples enable you to use the WebSphere MQ JMS ASF in a standalone environment (that is, you do not need a suitable application server). Also, they provide examples of how to implement these interfaces and take advantage of the WebSphere MQ JMS ASF. These examples are intended to aid both WebSphere MQ JMS users, and vendors of other application servers.

**MyServerSession.java**

This class implements the javax.jms.ServerSession interface. It associates a thread with a JMS session. Instances of this class are pooled by a ServerSessionPool (see "MyServerSessionPool.java"). As a ServerSession, it must implement the following two methods:

- `getSession()`, which returns the JMS Session associated with this ServerSession
- `start()`, which starts this ServerSession's thread and results in the JMS Session's run() method being invoked

MyServerSession also implements the Runnable interface. Therefore, the creation of the ServerSession's thread can be based on this class, and does not need a separate class.

The class uses a `wait()-notify()` mechanism that is based on the values of two boolean flags, `ready` and `quit`. This mechanism means that the ServerSession creates and starts its associated thread during its construction. However, it does not automatically execute the body of the `run()` method. The body of the `run()` method is executed only when the `ready` flag is set to true by the `start()` method. The ASF calls the `start()` method when it is necessary to deliver messages to the associated JMS session.

For delivery, the `run()` method of the JMS session is called. The WebSphere MQ JMS ASF will have already loaded the `run()` method with messages.

After delivery completes, the `ready` flag is reset to false, and the owning ServerSessionPool is notified that delivery is complete. The ServerSession then remains in a wait state until either the `start()` method is called again, or the `close()` method is invoked and ends this ServerSession's thread.

**MyServerSessionPool.java**

This class implements the javax.jms.ServerSessionPool interface, creating and controlling access to a pool of ServerSessions.

In this implementation, the pool consists of a static array of ServerSession objects that are created during the construction of the pool. The following four parameters are passed into the constructor:

- `javax.jms.Connection connection`
  The connection used to create JMS sessions.
- `int capacity`
  The size of the array of MyServerSession objects.
Application server sample code

- int ackMode
  The required acknowledge mode of the JMS sessions.
- MessageListenerFactory mlf
  The MessageListenerFactory that creates the message listener that is supplied to the JMS sessions. See MessageListenerFactory.java.

The pool’s constructor uses these parameters to create an array of MyServerSession objects. The supplied connection is used to create JMS sessions of the given acknowledge mode and correct domain (QueueSessions for point-to-point and TopicSessions for publish/subscribe). The sessions are supplied with a message listener. Finally, the ServerSession objects, based on the JMS sessions, are created.

This sample implementation is a static model. That is, all the ServerSessions in the pool are created when the pool is created, and after this the pool cannot grow or shrink. This approach is just for simplicity. It is possible for a ServerSessionPool to use a sophisticated algorithm to create ServerSessions dynamically, as needed.

MyServerSessionPool keeps a record of which ServerSessions are currently in use by maintaining an array of boolean values called inUse. These booleans are all initialized to false. When the getServerSession method is invoked and requests a ServerSession from the pool, the inUse array is searched for the first false value. When one is found, the boolean is set to true and the corresponding ServerSession is returned. If there are no false values in the inUse array, the getServerSession method must wait() until notification occurs.

Notification occurs in either of the following circumstances:
- The pool’s close() method is called, indicating that the pool must be shut down.
- A ServerSession that is currently in use completes its workload and calls the serverSessionFinished method. The serverSessionFinished method returns the ServerSession to the pool, and sets the corresponding inUse flag to false. The ServerSession then becomes eligible for reuse.

MessageListenerFactory.java
In this sample, a message listener factory object is associated with each ServerSessionPool instance. The MessageListenerFactory class represents a very simple interface that is used to obtain an instance of a class that implements the javax.jms.MessageListener interface. The class contains a single method:

javax.jms.MessageListener createMessageListener();

An implementation of this interface is supplied when the ServerSessionPool is constructed. This object is used to create message listeners for the individual JMS sessions that back up the ServerSessions in the pool. This architecture means that each separate implementation of the MessageListenerFactory interface must have its own ServerSessionPool.

WebSphere MQ JMS includes a sample MessageListenerFactory implementation, which is discussed in CountingMessageListenerFactory.java on page 288.
Examples of ASF use

There is a set of classes, with their source, in the directory
<install_dir>/samples/jms/asf (where <install_dir> is the installation directory
directory for WebSphere MQ JMS). These classes use the WebSphere MQ JMS application
server facilities that are described in “ASF classes and functions” on page 277,
within the sample standalone application server environment that is described in
“Application server sample code” on page 283.

These samples provide examples of ASF use from the perspective of a client
application:

- A simple point-to-point example uses:
  - ASFClient1.java
  - Load1.java
  - CountingMessageListenerFactory.java
- A more complex point-to-point example uses:
  - ASFClient2.java
  - Load2.java
  - CountingMessageListenerFactory.java
  - LoggingMessageListenerFactory.java
- A simple publish/subscribe example uses:
  - ASFClient3.java
  - TopicLoad.java
  - CountingMessageListenerFactory.java
- A more complex publish/subscribe example uses:
  - ASFClient4.java
  - TopicLoad.java
  - CountingMessageListenerFactory.java
  - LoggingMessageListenerFactory.java
- A publish/subscribe example using a durable ConnectionConsumer uses:
  - ASFClient5.java
  - TopicLoad.java

The following sections describe each class in turn.

**Load1.java**

This class is a generic JMS application that loads a given queue with a number of
messages, then terminates. It can either retrieve the required administered objects
from a JNDI namespace, or create them explicitly, using the WebSphere MQ JMS
classes that implement these interfaces. The administered objects that are required
are a QueueConnectionFactory and a queue. You can use the command line
options to set the number of messages with which to load the queue, and the sleep
time between individual message puts.

This application has two versions of the command line syntax.

For use with JNDI, the syntax is:

```
java Load1 [-icf jndiICF] [-url jndiURL] [-qcfLookup qcfLookup]
[-qLookup qLookup] [-sleep sleepTime] [-msgs numMsgs]
```
For use without JNDI, the syntax is:
```
java Load1 -nojndi [-qmgr qMgrName] [-q qName]
[-sleep sleepTime] [-msgs numMsgs]
```

Table 30 describes the parameters and gives their defaults.

### Table 30. Load1 parameters and defaults

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>jndiICF</td>
<td>Initial context factory class used for JNDI</td>
<td>com.sun.jndi.ldap.LdapCtxFactory</td>
</tr>
<tr>
<td>jndiURL</td>
<td>Provider URL used for JNDI</td>
<td>ldap://localhost/o=ibm,c=us</td>
</tr>
<tr>
<td>qcfLookup</td>
<td>JNDI lookup key used for QueueConnectionFactory</td>
<td>cn=qcf</td>
</tr>
<tr>
<td>qLookup</td>
<td>JNDI lookup key used for Queue</td>
<td>cn=q</td>
</tr>
<tr>
<td>qMgrName</td>
<td>Name of queue manager to connect to</td>
<td>&quot;&quot; (use the default queue manager)</td>
</tr>
<tr>
<td>qName</td>
<td>Name of queue to load</td>
<td>SYSTEM.DEFAULT.LOCAL.QUEUE</td>
</tr>
<tr>
<td>sleepTime</td>
<td>Time (in milliseconds) to pause between message puts</td>
<td>0 (no pause)</td>
</tr>
<tr>
<td>numMsgs</td>
<td>Number of messages to put</td>
<td>1000</td>
</tr>
</tbody>
</table>

If there are any errors, an error message is displayed and the application terminates.

You can use this application to simulate message load on a WebSphere MQ queue. In turn, this message load can trigger the ASF-enabled applications described in the following sections. The messages put to the queue are simple JMS TextMessage objects. These objects do not contain user-defined message properties, which could be useful to make use of different message listeners. The source code is supplied so that you can modify this load application if necessary.

### CountingMessageListenerFactory.java

This file contains definitions for two classes:
- CountingMessageListener
- CountingMessageListenerFactory

CountingMessageListener is a very simple implementation of the javax.jms.MessageListener interface. It keeps a record of the number of times its onMessage method has been invoked, but does nothing with the messages it is passed.

CountingMessageListenerFactory is the factory class for CountingMessageListener. It is an implementation of the MessageListenerFactory interface described in “MessageListenerFactory.java” on page 286. This factory keeps a record of all the message listeners that it produces. It also includes a method, printStats(), which displays usage statistics for each of these listeners.
ASFClient1.java

This application acts as a client of the WebSphere MQ JMS ASF. It sets up a single ConnectionConsumer to consume the messages in a single WebSphere MQ queue. It displays throughput statistics for each message listener that is used, and terminates after one minute.

The application can either retrieve the required administered objects from a JNDI namespace, or create them explicitly, using the WebSphere MQ JMS classes that implement these interfaces. The administered objects that are required are a QueueConnectionFactory and a queue.

This application has two versions of the command line syntax:

For use with JNDI, the syntax is:
```
```

For use without JNDI, the syntax is:
```
java ASFClient1 -nojndi [-qmgr qMgrName] [-q qName] [-poolSize poolSize] [-batchSize batchSize]
```

Table 31 describes the parameters and gives their defaults.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>jndiICF</td>
<td>Initial context factory class used for JNDI</td>
<td>com.sun.jndi.ldap.LdapCtxFactory</td>
</tr>
<tr>
<td>jndiURL</td>
<td>Provider URL used for JNDI</td>
<td>ldap://localhost/o=ibm,c=us</td>
</tr>
<tr>
<td>qcfLookup</td>
<td>JNDI lookup key used for QueueConnectionFactory</td>
<td>cn=qcf</td>
</tr>
<tr>
<td>qLookup</td>
<td>JNDI lookup key used for Queue</td>
<td>cn=q</td>
</tr>
<tr>
<td>qMgrName</td>
<td>Name of queue manager to connect to</td>
<td>&quot;&quot; (use the default queue manager)</td>
</tr>
<tr>
<td>qName</td>
<td>Name of queue to consume from</td>
<td>SYSTEM.DEFAULT.LOCAL.QUEUE</td>
</tr>
<tr>
<td>poolSize</td>
<td>The number of ServerSessions created in the ServerSessionPool being used</td>
<td>5</td>
</tr>
<tr>
<td>batchSize</td>
<td>The maximum number of message that can be assigned to a ServerSession at a time</td>
<td>10</td>
</tr>
</tbody>
</table>

The application obtains a QueueConnection from the QueueConnectionFactory.

A ServerSessionPool, in the form of a MyServerSessionPool, is constructed using:

- The QueueConnection that was created previously
- The required poolSize
- An acknowledge mode, AUTO_ACKNOWLEDGE
- An instance of a CountingMessageListenerFactory, as described in "CountingMessageListenerFactory.java" on page 288
The connection’s createConnectionConsumer method is invoked, passing in:

- The queue that was obtained earlier
- A null message selector (indicating that all messages should be accepted)
- The ServerSessionPool that was just created
- The batchSize that is required

The consumption of messages is then started by invoking the connection’s start() method.

The client application displays throughput statistics for each message listener that is used, displaying statistics every 10 seconds. After one minute, the connection is closed, the server session pool is stopped, and the application terminates.

**Load2.java**

This class is a JMS application that loads a given queue with a number of messages, then terminates, in a similar way to Load1.java. The command line syntax is also similar to that for Load1.java (substitute Load2 for Load1 in the syntax). For details, see "Load1.java on page 287.

The difference is that each message contains a user property called value, which takes a randomly selected integer value between 0 and 100. This property means that you can apply message selectors to the messages. Consequently, the messages can be shared between the two consumers that are created in the client application described in "ASFClient2.java".

**LoggingMessageListenerFactory.java**

This file contains definitions for two classes:

- LoggingMessageListener
- LoggingMessageListenerFactory

LoggingMessageListener is an implementation of the javax.jms.MessageListener interface. It takes the messages that are passed to it and writes an entry to the log file. The default log file is ./ASFClient2.log. You can inspect this file and check the messages that are sent to the connection consumer that is using this message listener.

LoggingMessageListenerFactory is the factory class for LoggingMessageListener. It is an implementation of the MessageListenerFactory interface described in "MessageListenerFactory.java" on page 286.

**ASFClient2.java**

ASFClient2.java is a slightly more complicated client application than ASFClient1.java. It creates two ConnectionConsumers that feed off the same queue, but that apply different message selectors. The application uses a CountingMessageListenerFactory for one consumer, and a LoggingMessageListenerFactory for the other. Use of two different message listener factories means that each consumer must have its own server session pool.

The application displays statistics that relate to one ConnectionConsumer on screen, and writes statistics that relate to the other ConnectionConsumer to a log file.
The command line syntax is similar to that for **ASFClient1.java** on page 289 (substitute ASFClient2 for ASFClient1 in the syntax). Each of the two server session pools contains the number of ServerSessions set by the poolSize parameter.

There should be an uneven distribution of messages. The messages loaded onto the source queue by Load2 contain a user property, where the value is between 0 and 100, evenly and randomly distributed. The message selector value>75 is applied to highConnectionConsumer, and the message selector value=75 is applied to normalConnectionConsumer. The highConnectionConsumer’s messages (approximately 25% of the total load) are sent to a LoggingMessageListener. The normalConnectionConsumer’s messages (approximately 75% of the total load) are sent to a CountingMessageListener.

When the client application runs, statistics that relate to the normalConnectionConsumer, and its associated CountingMessageListenerFactories, are printed to screen every 10 seconds. Statistics that relate to the highConnectionConsumer, and its associated LoggingMessageListenerFactories, are written to the log file.

You can inspect the screen and the log file to see the real destination of the messages. Add the totals for each of the CountingMessageListeners. As long as the client application does not terminate before all the messages are consumed, this accounts for approximately 75% of the load. The number of log file entries accounts for the remainder of the load. (If the client application terminates before all the messages are consumed, you can increase the application timeout.)

**TopicLoad.java**

This class is a JMS application that is a publish/subscribe version of the Load2 queue loader described in **Load2.java** on page 290. It publishes the required number of messages under the given topic, then terminates. Each message contains a user property called value, which takes a randomly selected integer value between 0 and 100.

To use this application, ensure that the broker is running and that the required setup is complete. For details, see "**Additional setup for publish/subscribe mode**" on page 26.

This application has two versions of the command line syntax.

For use with JNDI, the syntax is:

```
```

For use without JNDI, the syntax is:

```
java TopicLoad -nojndi [-qMgr qMgrName] [-t tName] [-sleep sleepTime] [-msgs numMsps]
```

**Table 32** describes the parameters and gives their defaults.

**Table 32. TopicLoad parameters and defaults**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>jndiICF</td>
<td>Initial context factory class used for JNDI</td>
<td>com.sun.jndi.ldap.LdapCfxFactory</td>
</tr>
<tr>
<td>jndiURL</td>
<td>Provider URL used for JNDI</td>
<td>ldap://localhost/o=ibm,c=us</td>
</tr>
</tbody>
</table>
Examples of ASF use

Table 32. TopicLoad parameters and defaults (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcfLookup</td>
<td>JNDI lookup key used for TopicConnectionFactory</td>
<td>cn=tcf</td>
</tr>
<tr>
<td>tLookup</td>
<td>JNDI lookup key used for Topic</td>
<td>cn=t</td>
</tr>
<tr>
<td>qMgrName</td>
<td>Name of queue manager to connect to, and broker queue manager to publish messages to</td>
<td>&quot;&quot; (use the default queue manager)</td>
</tr>
<tr>
<td>tName</td>
<td>Name of topic to publish to</td>
<td>MQJMS/ASF/TopicLoad</td>
</tr>
<tr>
<td>sleepTime</td>
<td>Time (in milliseconds) to pause between message puts</td>
<td>0 (no pause)</td>
</tr>
<tr>
<td>numMsgs</td>
<td>Number of messages to put</td>
<td>200</td>
</tr>
</tbody>
</table>

If there are any errors, an error message is displayed and the application terminates.

ASFClient3.java

ASFClient3.java is a client application that is a publish/subscribe version of "ASFClient1.java" on page 289. It sets up a single ConnectionConsumer to consume the messages published on a single Topic. It displays throughput statistics for each message listener that is used, and terminates after one minute.

This application has two versions of the command line syntax.

For use with JNDI, the syntax is:

```java
```

For use without JNDI, the syntax is:

```java
java ASFClient3 -nojndi [-qmgr qMgrName] [-t tName] [-poolsize poolSize] [-batchsize batchSize]
```

Table 33 describes the parameters and gives their defaults.

Table 33. ASFClient3 parameters and defaults

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>jndiICF</td>
<td>Initial context factory class used for JNDI</td>
<td>com.sun.jndi.LdapCtxFactory</td>
</tr>
<tr>
<td>jndiURL</td>
<td>Provider URL used for JNDI</td>
<td>ldap://localhost/o=ibm,c=us</td>
</tr>
<tr>
<td>tcfLookup</td>
<td>JNDI lookup key used for TopicConnectionFactory</td>
<td>cn=tcf</td>
</tr>
<tr>
<td>tLookup</td>
<td>JNDI lookup key used for Topic</td>
<td>cn=t</td>
</tr>
<tr>
<td>qMgrName</td>
<td>Name of queue manager to connect to, and broker queue manager to publish messages to</td>
<td>&quot;&quot; (use the default queue manager)</td>
</tr>
<tr>
<td>tName</td>
<td>Name of topic to consume from</td>
<td>MQJMS/ASF/TopicLoad</td>
</tr>
<tr>
<td>poolSize</td>
<td>The number of ServerSessions created in the ServerSessionPool being used</td>
<td>5</td>
</tr>
</tbody>
</table>
### ASFClient4.java

ASFClient4.java is a more complex publish/subscribe client application. It creates three ConnectionConsumers that all feed off the same topic, but each one applies different message selectors.

The first two consumers use *high* and *normal* message selectors, in the same way as described for the application "ASFClient2.java" on page 290. The third consumer does not use any message selector. The application uses two CountingMessageListenerFactories for the two selector-based consumers, and a LoggingMessageListenerFactory for the third consumer. Because the application uses different message listener factories, each consumer must have its own server session pool.

The application displays statistics that relate to the two selector-based consumers on screen. It writes statistics that relate to the third ConnectionConsumer to a log file.

The command line syntax is similar to that for "ASFClient3.java" on page 292 (substitute ASFClient4 for ASFClient3 in the syntax). Each of the three server session pools contains the number of ServerSessions set by the poolSize parameter.

When the client application runs, statistics that relate to the normalConnectionConsumer and the highConnectionConsumer, and their associated CountingMessageListenerFactories, are printed to screen every 10 seconds. Statistics that relate to the third ConnectionConsumer, and its associated LoggingMessageListenerFactories, are written to the log file.

You can inspect the screen and the log file to see the real destination of the messages. Add the totals for each of the CountingMessageListeners and inspect the number of log file entries.

The distribution of messages is different from the distribution obtained by a point-to-point version of the same application (ASFClient2.java). This is because, in the publish/subscribe domain, each consumer of a topic obtains its own copy of each message published on that topic. In this application, for a given topic load, the high and normal consumers receive approximately 25% and 75% of the load, respectively. The third consumer still receives 100% of the load. Therefore, the total number of messages received is greater than 100% of the load originally published on the topic.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>batchSize</td>
<td>The maximum number of message that can be assigned to a ServerSession at a time</td>
<td>10</td>
</tr>
</tbody>
</table>

Like ASFClient1, the client application displays throughput statistics for each message listener that is used, displaying statistics every 10 seconds. After one minute, the connection is closed, the server session pool is stopped, and the application terminates.

### Examples of ASF use

Table 33. ASFClient3 parameters and defaults (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>batchSize</td>
<td>The maximum number of message that can be assigned to a ServerSession at a time</td>
<td>10</td>
</tr>
</tbody>
</table>
ASFClient5.java

This sample exercises the durable publish/subscribe ConnectionConsumer functionality in WebSphere MQ JMS.

You invoke it with the same command-line options as the ASFClient4 sample, and, as with the other samples, the TopicLoad sample application can be used to trigger the consumer that is created. For details of TopicLoad, see “TopicLoad.java” on page 291.

When invoked, ASFClient5 displays a menu of three options:
1. Create/reactivate a durable ConnectionConsumer
2. Unsubscribe a durable ConnectionConsumer
X. Exit

If you choose option 1, and this is the first time this sample has been run, a new durable ConnectionConsumer is created using the given name. It then displays one minute’s worth of throughput statistics, rather like the other samples, before closing the connection and terminating.

Having created a durable consumer, messages published on the topic in question continues to arrive at the consumer’s destination even though the consumer is inactive.

This can be confirmed by running ASFClient5 again, and selecting option 1. This reactivates the named durable consumer, and the statistics displayed show that any relevant messages published during the period of inactivity were subsequently delivered to the consumer.

If you run ASFClient5 again and select option 2, this unsubscribes the named durable ConnectionConsumer and discards any outstanding messages delivered to it. Do this to ensure that the broker does not continue to deliver unwanted messages.
Chapter 15. JMS interfaces and classes

WebSphere MQ classes for Java Message Service consists of a number of Java classes and interfaces that are based on the Sun javax.jms package of interfaces and classes. Write your clients using the Sun interfaces and classes that are listed below, and that are described in detail in the following sections. The names of the WebSphere MQ objects that implement the Sun interfaces and classes have a prefix of MQ (unless stated otherwise in the object description). The descriptions include details about any deviations of the WebSphere MQ objects from the standard JMS definitions. These deviations are marked with *.

Sun Java Message Service classes and interfaces

The following tables list the JMS objects contained in the package javax.jms. Interfaces marked with * are implemented by applications. Interfaces marked with ** are implemented by application servers.

Table 34. Summary of interfaces in package javax.jms

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BytesMessage</td>
<td>Used to send a message containing a stream of uninterpreted bytes.</td>
</tr>
<tr>
<td>Connection</td>
<td>A client’s active connection to its JMS provider.</td>
</tr>
<tr>
<td>ConnectionConsumer</td>
<td>For application servers, a special facility for creating a ConnectionConsumer.</td>
</tr>
<tr>
<td>ConnectionFactory</td>
<td>A set of connection configuration parameters that an administrator has defined. In JMS 1.1 only, a client can use a ConnectionFactory to create a Connection to a JMS point-to-point provider, a JMS publish/subscribe provider, or both.</td>
</tr>
<tr>
<td>ConnectionMetaData</td>
<td>Information that describes the Connection.</td>
</tr>
<tr>
<td>DeliveryMode</td>
<td>Delivery modes supported by JMS.</td>
</tr>
<tr>
<td>Destination</td>
<td>Parent interface for Queue and Topic.</td>
</tr>
<tr>
<td>ExceptionListener*</td>
<td>Used to receive exceptions thrown by Connections asynchronous delivery threads.</td>
</tr>
<tr>
<td>MapMessage</td>
<td>Used to send a set of name-value pairs where names are Strings and values are Java primitive types.</td>
</tr>
<tr>
<td>Message</td>
<td>Root interface of all JMS messages.</td>
</tr>
<tr>
<td>MessageConsumer</td>
<td>In JMS 1.1, a client uses a MessageConsumer to receive messages from a Destination.</td>
</tr>
<tr>
<td>MessageListener*</td>
<td>Used to receive asynchronously delivered messages.</td>
</tr>
<tr>
<td>MessageProducer</td>
<td>Used by a client to send messages to a destination.</td>
</tr>
<tr>
<td>ObjectMessage</td>
<td>Used to send a message that contains a serializable Java object.</td>
</tr>
<tr>
<td>Queue</td>
<td>A provider-specific queue name.</td>
</tr>
<tr>
<td>QueueBrowser</td>
<td>Used by a client to look at messages on a queue without removing them.</td>
</tr>
<tr>
<td>QueueConnection</td>
<td>An active connection to a JMS point-to-point provider.</td>
</tr>
</tbody>
</table>
Table 34. Summary of interfaces in package `javax.jms` (continued)

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueueConnectionFactory</td>
<td>Used by a client to create <code>QueueConnections</code> with a JMS point-to-point provider.</td>
</tr>
<tr>
<td>QueueReceiver</td>
<td>Used by a client to receive messages that have been delivered to a queue.</td>
</tr>
<tr>
<td>QueueSender</td>
<td>Used by a client to send messages to a queue.</td>
</tr>
<tr>
<td>QueueSession</td>
<td>Provides methods to create <code>QueueReceivers</code>, <code>QueueSenders</code>, <code>QueueBrowsers</code> and <code>TemporaryQueues</code>.</td>
</tr>
<tr>
<td>ServerSession **</td>
<td>An object implemented by an application server.</td>
</tr>
<tr>
<td>ServerSessionPool **</td>
<td>An object implemented by an application server to provide a pool of <code>Session</code> for processing the messages of a <code>ConnectionConsumer</code>.</td>
</tr>
<tr>
<td>Session</td>
<td>A single-threaded context for producing and consuming messages.</td>
</tr>
<tr>
<td>StreamMessage</td>
<td>Used to send a stream of Java primitives.</td>
</tr>
<tr>
<td>TemporaryQueue</td>
<td>A unique queue object created for the duration of a <code>QueueConnection</code>.</td>
</tr>
<tr>
<td>TemporaryTopic</td>
<td>A unique Topic object created for the duration of a <code>TopicConnection</code>.</td>
</tr>
<tr>
<td>TextMessage</td>
<td>Used to send a message containing a <code>java.lang.String</code>.</td>
</tr>
<tr>
<td>Topic</td>
<td>A provider-specific topic name.</td>
</tr>
<tr>
<td>TopicConnection</td>
<td>An active connection to a JMS Publish/Subscribe provider.</td>
</tr>
<tr>
<td>TopicConnectionFactory</td>
<td>Used by a client to create <code>TopicConnections</code> with a JMS Publish/Subscribe provider.</td>
</tr>
<tr>
<td>TopicPublisher</td>
<td>Used by a client to publish messages on a topic.</td>
</tr>
<tr>
<td>TopicSession</td>
<td>Provides methods to create <code>TopicPublishers</code>, <code>TopicSubscribers</code> and <code>TemporaryTopics</code>.</td>
</tr>
<tr>
<td>TopicSubscriber</td>
<td>Used by a client to receive messages that have been published to a topic.</td>
</tr>
<tr>
<td>XAConnection</td>
<td>Extends the capability of <code>Connection</code> by providing an <code>XASession</code>.</td>
</tr>
<tr>
<td>XAConnectionFactory</td>
<td>Used by some application servers to provide support for grouping Java Transaction Service (JTS)-capable resource use into a distributed transaction.</td>
</tr>
<tr>
<td>XAQueueConnection</td>
<td>Provides the same create options as <code>QueueConnection</code>.</td>
</tr>
<tr>
<td>XAQueueConnectionFactory</td>
<td>Provides the same create options as a <code>QueueConnectionFactory</code>.</td>
</tr>
<tr>
<td>XAQueueSession</td>
<td>Provides a regular <code>QueueSession</code> that can be used to create <code>QueueReceivers</code>, <code>QueueSenders</code> and <code>QueueBrowsers</code>.</td>
</tr>
<tr>
<td>XASession</td>
<td>Extends the capability of <code>Session</code> by adding access to a JMS provider’s support for the Java Transaction API (JTA).</td>
</tr>
<tr>
<td>XATopicConnection</td>
<td>Provides the same create options as <code>TopicConnection</code>.</td>
</tr>
<tr>
<td>XATopicConnectionFactory</td>
<td>Provides the same create options as <code>TopicConnectionFactory</code>.</td>
</tr>
</tbody>
</table>
### Table 34. Summary of interfaces in package javax.jms (continued)

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XATopicSession</td>
<td>Provides a regular TopicSession which can be used to create TopicSubscribers and TopicPublishers.</td>
</tr>
</tbody>
</table>

### Table 35. Summary of classes in package javax.jms

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueueRequestor</td>
<td>A helper class to simplify making service requests.</td>
</tr>
<tr>
<td>TopicRequestor</td>
<td>A helper class to simplify making service requests.</td>
</tr>
</tbody>
</table>
WebSphere MQ JMS classes

Two packages contain the WebSphere MQ classes for Java Message Service that implement the Sun interfaces. Table 36 shows the interfaces implemented by classes in the `com.ibm.mq.jms` package; Table 37 on page 299 shows the interfaces implemented by classes in the `com.ibm.jms` package.

You do not usually use the implementation classes directly; you program to the JMS interfaces. Many of the interfaces do not apply when running a publish/subscribe application on a direct connection to the IBM WebSphere MQ Event Broker. Where the names of implementation classes are listed, provider-specific methods are documented in this chapter.

Table 36. Summary of classes in package `com.ibm.mq.jms`

<table>
<thead>
<tr>
<th>JMS interface</th>
<th>Client or bindings implementation</th>
<th>Direct connection to WebSphere MQ Event Broker implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>MQConnection</td>
<td>Y</td>
</tr>
<tr>
<td>ConnectionConsumer</td>
<td>MQConnectionConsumer</td>
<td></td>
</tr>
<tr>
<td>ConnectionFactory</td>
<td>MQConnectionFactory</td>
<td>Y</td>
</tr>
<tr>
<td>ConnectionMetaData</td>
<td>MQConnectionMetaData</td>
<td>Y</td>
</tr>
<tr>
<td>Destination</td>
<td>MQDestination</td>
<td></td>
</tr>
<tr>
<td>MessageConsumer</td>
<td>MQMessageConsumer</td>
<td></td>
</tr>
<tr>
<td>MessageProducer</td>
<td>MQMessageProducer</td>
<td></td>
</tr>
<tr>
<td>Queue</td>
<td>MQQueue</td>
<td></td>
</tr>
<tr>
<td>QueueBrowser</td>
<td>MQQueueBrowser</td>
<td></td>
</tr>
<tr>
<td>QueueConnection</td>
<td>MQQueueConnection</td>
<td></td>
</tr>
<tr>
<td>QueueConnectionFactory</td>
<td>MQQueueConnectionFactory</td>
<td></td>
</tr>
<tr>
<td>QueueEnumeration</td>
<td>MQQueueEnumeration</td>
<td></td>
</tr>
<tr>
<td>QueueReceiver</td>
<td>MQQueueReceiver</td>
<td></td>
</tr>
<tr>
<td>QueueSender</td>
<td>MQQueueSender</td>
<td></td>
</tr>
<tr>
<td>QueueSession</td>
<td>MQQueueSession</td>
<td></td>
</tr>
<tr>
<td>Session</td>
<td>MQSession</td>
<td>Y</td>
</tr>
<tr>
<td>TemporaryQueue</td>
<td>MQTemporaryQueue</td>
<td></td>
</tr>
<tr>
<td>TemporaryTopic</td>
<td>MQTemporaryTopic</td>
<td>Y</td>
</tr>
<tr>
<td>Topic</td>
<td>MQTopic</td>
<td>Y</td>
</tr>
<tr>
<td>TopicConnection</td>
<td>MQTopicConnection</td>
<td>Y</td>
</tr>
<tr>
<td>TopicConnectionFactory</td>
<td>MQTopicConnectionFactory</td>
<td>Y</td>
</tr>
<tr>
<td>TopicPublisher</td>
<td>MQTopicPublisher</td>
<td>Y</td>
</tr>
<tr>
<td>TopicSession</td>
<td>MQTopicSession</td>
<td>Y</td>
</tr>
<tr>
<td>TopicSubscriber</td>
<td>MQTopicSubscriber</td>
<td>Y</td>
</tr>
<tr>
<td>XAConnection</td>
<td>MQXAConnection</td>
<td></td>
</tr>
<tr>
<td>XAConnectionFactory</td>
<td>MQXAConnectionFactory</td>
<td></td>
</tr>
<tr>
<td>XAQueueConnection</td>
<td>MQXAQueueConnection</td>
<td></td>
</tr>
</tbody>
</table>
Table 36. Summary of classes in package com.ibm.mq.jms (continued)

<table>
<thead>
<tr>
<th>JMS interface</th>
<th>Client or bindings implementation</th>
<th>Direct connection to WebSphere MQ Event Broker implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>XAQueueConnectionFactory</td>
<td>MQXAQueueConnectionFactory</td>
<td></td>
</tr>
<tr>
<td>XAQueueSession</td>
<td>MQXAQueueSession</td>
<td></td>
</tr>
<tr>
<td>XASession</td>
<td>MQXASession</td>
<td></td>
</tr>
<tr>
<td>XATopicConnectionFactory</td>
<td>MQXATopicConnectionFactory</td>
<td></td>
</tr>
<tr>
<td>XATopicSession</td>
<td>MQXATopicSession</td>
<td></td>
</tr>
</tbody>
</table>

Table 37. Summary of classes in package com.ibm.jms

<table>
<thead>
<tr>
<th>JMS interface</th>
<th>Client or bindings implementation</th>
<th>Direct connection to WebSphere MQ Event Broker implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BytesMessage</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MapMessage</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Message</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>ObjectMessage</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>StreamMessage</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>TextMessage</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

A sample implementation of the following JMS interfaces is supplied in the WebSphere MQ classes for Java Message Service.
- ServerSession
- ServerSessionPool

See “Application server sample code” on page 283 for more information.
BytesMessage

public interface BytesMessage
extends Message

WebSphere MQ class: JMSBytesMessage

java.lang.Object
  +----com.ibm.jms.JMSMessage
    +----com.ibm.jms.JMSBytesMessage

Use a BytesMessage to send a message containing a stream of uninterpreted bytes. It inherits Message and adds a bytes message body. The receiver of the message supplies the interpretation of the bytes.

Note: This message type is for client encoding of existing message formats. If possible, use one of the other self-defining message types instead.

See also: MapMessage, Message, ObjectMessage, StreamMessage, and TextMessage

Methods

getBodyLength (JMS 1.1 only)

public long getBodyLength() throws JMSException

Get the number of bytes in the message body when the message is in read-only mode. The value returned can be used to allocate a byte array. The value 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

Throws:

* JMSException if JMS fails to read the message because of an internal JMS error.
* MessageNotReadableException if the message is in write-only mode.

readBoolean

public boolean readBoolean() throws JMSException

Read a boolean from the bytes message.

Returns:
The boolean value read.

Throws:

* MessageNotReadableException if the message is in write-only mode.
* JMSException if JMS fails to read the message because of an internal JMS error.
* MessageEOFException if it is the end of the message bytes.
**readByte**

```java
class BytesMessage {
    public byte readByte() throws JMSException {
        // implementation...
    }
}
```

**readByte**

```java
class BytesMessage {
    public int readBytes(byte[] value) throws JMSException {
        // implementation...
    }
}
```

**readBytes**

```java
class BytesMessage {
    public int readBytes(byte[] value, int length) throws JMSException {
        // implementation...
    }
}
```

**readByte**

```java
class BytesMessage {
    public byte readByte() throws JMSException {
        Read a signed 8-bit value from the bytes message.

**Returns:**

The next byte from the bytes message as a signed 8-bit byte.

**Throws:**

- MessageNotReadableException if the message is in write-only mode.
- MessageEOFException if it is the end of the message bytes.
- JMSException if JMS fails to read the message because of an internal JMS error.

**readBytes**

```java
class BytesMessage {
    public int readBytes(byte[] value) throws JMSException {
        Read a byte array from the bytes message. If there are sufficient bytes remaining in the stream, the entire buffer is filled; if not, the buffer is partially filled.

**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 bytes has been reached.

**Throws:**

- MessageNotReadableException if the message is in write-only mode.
- JMSException if JMS fails to read the message because of an internal JMS error.

**readBytes**

```java
class BytesMessage {
    public int readBytes(byte[] value, int length) throws JMSException {
        Read a portion of the bytes message.

**Parameters:**

- value: the buffer into which the data is read.
- length: the number of bytes to read.

**Returns:**

The total number of bytes read into the buffer, or -1 if there is no more data because the end of the bytes has been reached.

**Throws:**

- MessageNotReadableException if the message is in write-only mode.
- IndexOutOfBoundsException if length is negative, or is less than the length of the array value
- JMSException if JMS fails to read the message because of an internal JMS error.
BytesMessage

readChar
public char readChar() throws JMSException

Read a Unicode character value from the bytes message.

Returns:
The next two bytes from the bytes message as a Unicode character.

Throws:
• MessageNotReadableException if the message is in write-only mode.
• MessageEOFException if it is the end of the message bytes.
• JMSException if JMS fails to read the message because of an internal JMS error.

readDouble
public double readDouble() throws JMSException

Read a double from the bytes message.

Returns:
The next eight bytes from the bytes message, interpreted as a double.

Throws:
• MessageNotReadableException if the message is in write-only mode.
• MessageEOFException if it is the end of the message bytes.
• JMSException if JMS fails to read the message because of an internal JMS error.

readFloat
public float readFloat() throws JMSException

Read a float from the bytes message.

Returns:
The next four bytes from the bytes message, interpreted as a float.

Throws:
• MessageNotReadableException if the message is in write-only mode.
• MessageEOFException if it is the end of the message bytes.
• JMSException if JMS fails to read the message because of an internal JMS error.

readInt
public int readInt() throws JMSException

Read a signed 32-bit integer from the bytes message.

Returns:
The next four bytes from the bytes message, interpreted as an int.

Throws:
• MessageNotReadableException if the message is in write-only mode.
• MessageEOFException if it is the end of the message bytes.
BytesMessage

- JMSException if JMS fails to read the message because of an internal JMS error.

**readLong**

```java
public long readLong() throws JMSException
```

Read a signed 64-bit integer from the bytes message.

**Returns:**
- The next eight bytes from the bytes message, interpreted as a long.

**Throws:**
- MessageNotReadableException if the message is in write-only mode.
- MessageEOFException if it is the end of the message bytes.
- JMSException if JMS fails to read the message because of an internal JMS error.

**readShort**

```java
public short readShort() throws JMSException
```

Read a signed 16-bit number from the bytes message.

**Returns:**
- The next two bytes from the bytes message, interpreted as a signed 16-bit number.

**Throws:**
- MessageNotReadableException if the message is in write-only mode.
- MessageEOFException if it is the end of the message bytes.
- JMSException if JMS fails to read the message because of an internal JMS error.

**readUnsignedByte**

```java
public int readUnsignedByte() throws JMSException
```

Read an unsigned 8-bit number from the bytes message.

**Returns:**
- The next byte from the bytes message, interpreted as an unsigned 8-bit number.

**Throws:**
- MessageNotReadableException if the message is in write-only mode.
- MessageEOFException if it is the end of the message bytes.
- JMSException if JMS fails to read the message because of an internal JMS error.
BytesMessage

readUnsignedShort

public int readUnsignedShort() throws JMSException

Read an unsigned 16-bit number from the bytes message.

Returns:
The next two bytes from the bytes message, interpreted as an unsigned 16-bit integer.

Throws:
• MessageNotReadableException if the message is in write-only mode.
• MessageEOFException if it is the end of the message bytes.
• JMSException if JMS fails to read the message because of an internal JMS error.

readUTF

public java.lang.String readUTF() throws JMSException

Read a string that has been encoded using a modified UTF-8 format from the bytes message. The first two bytes are interpreted as a 2-byte length field.

Returns:
A Unicode string from the bytes message.

Throws:
• MessageNotReadableException if the message is in write-only mode.
• MessageEOFException if it is the end of the message bytes.
• JMSException if JMS fails to read the message because of an internal JMS error.

reset

public void reset() throws JMSException

Put the message body in read-only mode, and reposition the stream of bytes to the beginning.

Throws:
• JMSException if JMS fails to reset the message because of an internal JMS error.
• MessageFormatException if message has an incorrect format

writeBoolean

public void writeBoolean(boolean value) throws JMSException

Write a boolean to the bytes message as a 1-byte value. The value true is written out as the value (byte)1; the value false is written out as the value (byte)0.

Parameters:
value: the boolean value to be written.

Throws:
• MessageNotWriteableException if message in read-only mode.
• JMSException if JMS fails to write the message because of an internal JMS error.
writeByte
public void writeByte(byte value) throws JMSException

Write a byte to the bytes message as a 1-byte value.

Parameters:
  value: the byte value to be written.

Throws:
  • MessageNotWritableException if message in read-only mode.
  • JMSException if JMS fails to write the message because of an internal JMS error.

writeBytes
public void writeBytes(byte[] value) throws JMSException

Write a byte array to the bytes message.

Parameters:
  value: the byte array to be written.

Throws:
  • MessageNotWritableException if message in read-only mode.
  • JMSException if JMS fails to write the message because of an internal JMS error.

writeBytes
public void writeBytes(byte[] value, int offset, int length) throws JMSException

Write a portion of a byte array to the bytes message.

Parameters:
  • value: the byte array value to be written.
  • offset: the initial offset within the byte array.
  • length: the number of bytes to use.

Throws:
  • MessageNotWritableException if message in read-only mode.
  • JMSException if JMS fails to write the message because of an internal JMS error.

writeChar
public void writeChar(char value) throws JMSException

Write a char to the bytes message as a 2-byte value, high byte first.

Parameters:
  value: the char value to be written.

Throws:
  • MessageNotWritableException if message in read-only mode.
  • JMSException if JMS fails to write the message because of an internal JMS error.
writeDouble

public void writeDouble(double value) throws JMSException

Convert the double argument to a long using doubleToLongBits method in class Double, and then write that long value to the bytes message as an 8-byte quantity.

Parameters:
value: the double value to be written.

Throws:
• MessageNotWriteableException if message in read-only mode.
• JMSException if JMS fails to write the message because of an internal JMS error.

writeFloat

public void writeFloat(float value) throws JMSException

Convert the float argument to an int using floatToIntBits method in class Float, and then write that int value to the bytes message as a 4-byte quantity.

Parameters:
value: the float value to be written.

Throws:
• MessageNotWriteableException - if message in read-only mode.
• JMSException if JMS fails to write the message because of an internal JMS error.

writeInt

public void writeInt(int value) throws JMSException

Write an int to the bytes message as four bytes.

Parameters:
value: the int to be written.

Throws:
• MessageNotWriteableException if message in read-only mode.
• JMSException if JMS fails to write the message because of an internal JMS error.

writeLong

public void writeLong(long value) throws JMSException

Write a long to the bytes message as eight bytes.

Parameters:
value: the long to be written.

Throws:
• MessageNotWriteableException if message in read-only mode.
• JMSException if JMS fails to write the message because of an internal JMS error.
writeObject

    public void writeObject(java.lang.Object value)
    throws JMSException

Write a Java object to the bytes message.

**Note:** This method works only for the primitive object types (such as Integer, Double, and Long), Strings, and byte arrays.

**Parameters:**
- `value`: the Java object to be written.

**Throws:**
- `MessageNotWriteableException` if message in read-only mode.
- `MessageFormatException` if object is not a valid type.
- `JMSException` if JMS fails to write the message because of an internal JMS error.

writeShort

    public void writeShort(short value) throws JMSException

Write a short to the bytes message as two bytes.

**Parameters:**
- `value`: the short to be written.

**Throws:**
- `MessageNotWriteableException` if message in read-only mode.
- `JMSException` if JMS fails to write the message because of an internal JMS error.

writeUTF

    public void writeUTF(java.lang.String value)
    throws JMSException

Write a string to the bytes message using UTF-8 encoding in a machine-independent manner. The UTF-8 string written to the buffer starts with a 2-byte length field.

**Parameters:**
- `value`: the String value to be written.

**Throws:**
- `MessageNotWriteableException` if message in read-only mode.
- `JMSException` if JMS fails to write the message because of an internal JMS error.
public class Cleanup implements Runnable

WebSphere MQ class: Cleanup

Cleanup contains utilities for dealing with broken non-durable subscriptions using the SUBSTATE(BROKER) option. It is not applicable if you use a direct connection to WebSphere MQ Event Broker.

See also: ConnectionFactory.

### WebSphere MQ constructor

**Cleanup**

```java
public Cleanup()

Default constructor.
```

```java
public Cleanup(MQTopicConnectionFactory mqtcf) throws JMSException

Constructor that copies property values from the supplied MQTopicConnectionFactory.
```

### Methods

**cleanup**

```java
public void cleanup() throws JMSException

Executes Cleanup once. If cleanupLevel is NONE, throws an IllegalStateException.
```

**getCCSID**

```java
public int getCCSID()

Get the character set of the queue manager.
```

**getChannel**

```java
public String getChannel()

For client only, get the channel that was used.
```

**getCleanupInterval**

```java
public long getCleanupInterval()

Retrieve the cleanup interval.
```
getCleanupLevel
   public int getCleanupLevel()

   Retrieve the cleanup level.

getExceptionListener
   public ExceptionListener getExceptionListener()

   Return the ExceptionListener.

getHostName
   public String getHostName()

   Retrieve the name of the host.

getPort
   public int getPort()

   For client connections, get the port number.

getQueueManager
   public String getQueueManager()

   Get the name of the queue manager.

getReceiveExit
   public String getReceiveExit()

   Get the name of the receive exit class.

getReceiveExitInit
   public String getReceiveExitInit()

   Get the initialization string that was passed to the receive exit class.

getSecurityExit
   public String getSecurityExit()

   Get the name of the security exit class.

getSecurityExitInit
   public String getSecurityExitInit()

   Get the security exit initialization string.

getSendExit
   public String getSendExit()

   Get the name of the send exit class.
Cleanup

getSendExitInit
public String getSendExitInit()

Get the send exit initialization string.

getTransportType
public int getTransportType()

Retrieve the transport type.

isRunning
public boolean isRunning()

Return true if the run() method is currently active.

main
public static void main(String args[])
	throws java.io.UnsupportedEncodingException

Invoke the utility from a command line. For details of the invocation options and parameters, see “Manual cleanup” on page 232. For information specific to JMS 1.1, see “Manual cleanup” on page 250.

run
public void run()

Run this utility in the background at intervals, as determined by the cleanupLevel and cleanupInterval properties.

setCCSID
public void setCCSID(int x) throws JMSException

Set the character set to be used when connecting to the queue manager. See Table 13 on page 127 for a list of allowed values. We recommend that you use the default value (819) for most situations.

setChannel
public void setChannel(String x) throws JMSException

For client only, set the channel to use.

setCleanupInterval
public void setCleanupInterval(long interval) throws JMSException

Set the cleanupInterval.

Parameters:
- interval: length of time in milliseconds between runs of the cleanup utility

Throws:
- JMSEException if interval is less than 0
setCleanupLevel

public void setCleanupLevel(int level) throws JMSException

Set the cleanup level to use. It can be one of

JMSC.MQJMS_CLEANUP_NONE
JMSC.MQJMS_CLEANUP_SAFE
JMSC.MQJMS_CLEANUP_STRONG
JMSC.MQJMS_CLEANUP_FORCE
JMSC.MQJMS_CLEANUP_NONDUR

setExceptionListener

public void setExceptionListener(ExceptionListener el)

Set the ExceptionListener. If set, the ExceptionListener receives any exceptions caused during the run() method. Shortly after issuing the exception to the ExceptionListener, Cleanup terminates.

setHostName

public void setHostName(String hostname)

For client connections, the name of the host to connect to.

setPort

public void setPort(int port) throws JMSException

Set the port for a client connection.

Parameters:

port: the new value to use.

Throws:

JMSException if the port is negative.

setQueueManager

public void setQueueManager(String x) throws JMSException

Set the name of the queue manager to connect to.

setReceiveExit

public void setReceiveExit(String receiveExit)

The name of a class that implements a receive exit.

setReceiveExitInit

public void setReceiveExitInit(String x)

Initialization string that is passed to the constructor of the receive exit class.
Cleanup

setSecurityExit
    public void setSecurityExit(String securityExit)

    The name of a class that implements a security exit.

setSecurityExitInit
    public void setSecurityExitInit(String x)

    Initialization string that is passed to the security exit constructor.

setSendExit
    public void setSendExit(String sendExit)

    The name of a class that implements a send exit.

setSendExitInit
    public void setSendExitInit(String x)

    Initialization string that is passed to the constructor of send exit.

setTransportType
    public void setTransportType(int x) throws JMSException

    Set the transport type to use. It can be one of the following:
    JMSC.MQJMS_TP_BINDINGS_MQ
    JMSC.MQJMS_TP_CLIENT_MQ_TCPIP

stop
    public void stop()

    Stop any currently running cleanup thread. Return when cleanup has
    finished. Do nothing if cleanup is not running.
Connection

public interface Connection

Subinterfaces: QueueConnection, TopicConnection, XAConnection, XAQueueConnection, and XATopicConnection

WebSphere MQ class: MQConnection

java.lang.Object

A JMS Connection is a client’s active connection to its JMS provider.

See also: ConnectionFactory, QueueConnection, and TopicConnection

Methods

close

public void close() throws JMSException

Because a provider can allocate some resources outside the JVM on behalf of a Connection, clients must close them when they are not needed. You cannot rely on garbage collection to reclaim these resources eventually, because this might not occur soon enough. There is no need to close the sessions, producers, and consumers of a closed connection.

Closing a connection causes any of its sessions’ in-process transactions to be rolled back. If a session’s work is coordinated by an external transaction manager, when using XASession, a session’s commit and rollback methods are not used and the result of a closed session’s work is determined later by a transaction manager. Closing a connection does not force an acknowledgement of client acknowledged sessions.

WebSphere MQ JMS keeps a pool of WebSphere MQ hConns available for use by sessions. Under some circumstances, Connection.close() clears this pool. If an application uses multiple connections sequentially, you can force the pool to remain active between JMS connections. To do this, register an MQPoolToken with com.ibm.mq.MQEnvironment for the lifetime of your JMS application. For details, see “Connection pooling” on page 80 and “MQEnvironment” on page 110.

Throws:

JMSException if the JMS implementation fails to close the connection because of an internal error. Examples are a failure to release resources or to close a socket connection.
createConnectionConsumer (JMS 1.1 only)
public ConnectionConsumer createConnectionConsumer
(Destination destination,
java.lang.String messageSelector,
ServerSessionPool sessionPool,
int maxMessages) throws JMSException

Create a connection consumer for this connection. This is an expert facility that is not used by regular JMS clients.

Parameters:
• destination: the destination to access.
• messageSelector: deliver only those messages with properties that match the message selector expression. A value of null or an empty string indicates that there is no message selector for the 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.

Throws:
• JMSException if the connection fails to create a connection consumer because of an internal JMS error, or because of incorrect arguments for sessionPool and messageSelector.
• InvalidDestinationException if the destination is not valid.
• InvalidSelectorException if the message selector is not valid.

See also:
ConnectionConsumer
createDurableConnectionConsumer (JMS 1.1 only)
public ConnectionConsumer createDurableConnectionConsumer
(Topic topic,
java.lang.String subscriptionName,
java.lang.String messageSelector,
ServerSessionPool sessionPool,
int maxMessages) throws JMSException

Create a durable connection consumer for this connection. This is an expert facility that is not used by regular JMS clients.

Note
For a direct connection to WebSphere MQ Event Broker, WebSphere Business Integration Event Broker, or WebSphere Business Integration Message Broker, this method throws a JMSException.

Parameters:
• topic: the topic to access.
• subscriptionName: the name of the durable subscription.
Connection

- `messageSelector`: deliver only those messages with properties that match the message selector expression. A value of null or an empty string indicates that there is no message selector for the message consumer.
- `sessionPool`: the server session pool to associate with this durable connection consumer.
- `maxMessages`: the maximum number of messages that can be assigned to a server session at one time.

Returns:
The durable connection consumer.

Throws:
- `JMSException` if the connection fails to create a connection consumer because of an internal JMS error, or because of incorrect arguments for `sessionPool` and `messageSelector`.
- `InvalidDestinationException` if the destination is not valid.
- `InvalidSelectorException` if the message selector is not valid.

See also:
- `ConnectionConsumer`

`createSession` (JMS 1.1 only)

```java
public Session createSession(boolean transacted,
                              int acknowledgeMode) throws JMSException
```

Create a session.

Parameters:
- `transacted`: if true, the session is transacted.
- `acknowledgeMode`: indicates whether the consumer or the client acknowledges any messages it receives. Possible values are:
  - `Session.AUTO_ACKNOWLEDGE`
  - `Session.CLIENT_ACKNOWLEDGE`
  - `Session.DUPS_OK_ACKNOWLEDGE`

This parameter is ignored if the session is transacted.

Returns:
A newly created session.

Throws:
- `JMSException` if the connection fails to create a session because of an internal JMS error, or because of lack of support for the specific transaction and acknowledgement mode.

See also:
- `Session.AUTO_ACKNOWLEDGE`,
- `Session.CLIENT_ACKNOWLEDGE`,
- `Session.DUPS_OK_ACKNOWLEDGE`
**getClientID**

public java.lang.String getClientID() throws JMSException

Get the client identifier for this connection. The client identifier can either be preconfigured by the administrator in a ConnectionFactory, or assigned by calling setClientID.

**Returns:**

The unique client identifier.

**Throws:**

JMSException if the JMS implementation fails to return the client ID for this connection because of an internal error.

**getExceptionListener**

public ExceptionListener getExceptionListener() throws JMSException

Get the ExceptionListener for this connection.

**Returns:**

The ExceptionListener for this connection.

**Throws:**

JMSException general exception if the JMS implementation fails to get the exception listener for this connection.

**getMetaData**

public ConnectionMetaData getMetaData() throws JMSException

Get the metadata for this connection.

**Returns:**

The connection metadata.

**Throws:**

JMSException general exception if the JMS implementation fails to get the connection metadata for this connection.

See also: "ConnectionMetaData" on page 335

**setClientID**

public void setClientID(java.lang.String clientID) throws JMSException

Set the client identifier for this connection.

**Note:** The client identifier is ignored for point-to-point connections.

---

**WebSphere MQ Event Broker note**

This method always throws an IllegalStateException when you make a direct connection to WebSphere MQ Event Broker.

**Parameters:**

clientID: the unique client identifier.

**Throws:**
Connection

- JMSException if the JMS implementation fails to set the client ID for this Connection because of an internal error.
- InvalidClientIDException if the JMS client specifies a non valid or duplicate client ID.
- IllegalStateException if attempting to set a connection’s client identifier at the wrong time, or if it has been configured administratively.

`setExceptionListener`

```java
public void setExceptionListener(ExceptionListener listener) throws JMSException
```

Set an exception listener for this connection.

**Parameters:**
- `listener`: the exception listener.

**Throws:**
- JMSException general exception if the JMS implementation fails to set the exception listener for this connection.

`start`

```java
public void start() throws JMSException
```

Start (or restart) a connection’s delivery of incoming messages. Starting a started session is ignored. Use the `stop` method to stop delivery.

**Throws:**
- JMSException if the JMS implementation fails to start the message delivery because of an internal error.

`stop`

```java
public void stop() throws JMSException
```

Used to stop a connection’s delivery of incoming messages temporarily. It can be restarted using its `start` method. When stopped, delivery to all the connection’s message consumers is inhibited. Synchronous receives are blocked, and messages are not delivered to message listeners.

Stopping a session has no affect on its ability to send messages. Stopping a stopped session is ignored.

**Throws:**
- JMSException if the JMS implementation fails to stop the message delivery because of an internal error.
public interface ConnectionConsumer

WebSphere MQ class: MQConnectionConsumer

For application servers, Connections provide a special facility to create a ConnectionConsumer. A Destination and a Property Selector specify the messages that it is to consume. Also, a ConnectionConsumer must be given a ServerSessionPool to use to process its messages.

See also: QueueConnection, and TopicConnection.

Methods

close()

public void close() throws JMSException

Because a provider can allocate some resources outside the JVM on behalf of a ConnectionConsumer, clients must close them when they are not needed. You cannot rely on garbage collection to reclaim these resources eventually, because this might not occur soon enough.

Throws:
   JMSException if a JMS implementation fails to release resources on behalf of ConnectionConsumer, or if it fails to close the connection consumer.

getServerSessionPool()

public ServerSessionPool getServerSessionPool() throws JMSException

Get the server session associated with this connection consumer.

Returns:
   The server session pool used by this connection consumer.

Throws:
   JMSException if a JMS implementation fails to get the server session pool associated with this connection consumer because of an internal error.
ConnectionFactory

public interface ConnectionFactory
Subinterfaces: QueueConnectionFactory, TopicConnectionFactory, XAQueueConnectionFactory, and XATopicConnectionFactory

WebSphere MQ class: MQConnectionFactory

java.lang.Object
   +----com.ibm.mq.jms.MQConnectionFactory

A ConnectionFactory encapsulates a set of connection configuration parameters that has been defined by an administrator. A client uses it to create a Connection with a JMS provider.

Note
For direct connections to WebSphere MQ Event Broker, WebSphere Business Integration Event Broker, or WebSphere Business Integration Message Broker, properties accessed by methods marked with a § are ignored.

See also: Connection, QueueConnectionFactory, and TopicConnectionFactory

WebSphere MQ constructor

MQConnectionFactory
   public MQConnectionFactory()

Methods

createConnection (JMS 1.1 only)
   public Connection createConnection() throws JMSException

Create a connection with the default user identity. The connection is created in stopped mode. No messages are delivered until the Connection.start method is called explicitly.

Returns:
A newly created connection.

Throws:
   • JMSException if JMS fails to create the connection because of an internal JMS error.
   • JMSSecurityException if client authentication fails because the user name or password is not valid.
ConnectionFactory

createConnection (JMS 1.1 only)

    public Connection createConnection(java.lang.String userName,
            java.lang.String password)
    throws JMSException

Create a connection with the specified user identity. The connection is
created in stopped mode. No messages are delivered until the
Connection.start method is called explicitly.

Parameters:
    • userName: the user name of the caller.
    • password: the password of the caller.

Returns:
    A newly created connection.

Throws:
    • JMSException if JMS fails to create the connection because of an
      internal JMS error.
    • JMSSecurityException if client authentication fails because the
      user name or password is not valid.

getBrokerCCSubQueue * §

    public String getBrokerCCSubQueue()

Get method for brokerCCSubQueue attribute.

Returns:
    The name of the nondurable subscription queue to use for a
    connection consumer.

getBrokerControlQueue * §

    public String getBrokerControlQueue()

Get method for brokerControlQueue attribute.

Returns:
    The broker’s control queue name

getBrokerPubQueue * §

    public String getBrokerPubQueue()

Get method for brokerPubQueue attribute.

Returns:
    The broker’s publish queue name.

getBrokerQueueManager * §

    public String getBrokerQueueManager()

Get method for brokerQueueManager attribute.

Returns:
    The broker’s queue manager name.
getBrokerSubQueue * §
    public String getBrokerSubQueue()
    Get method for brokerSubQueue attribute.
    Returns:
    The name of the nondurable subscription queue to use.

getBrokerVersion *
    public int getBrokerVersion()
    Get method for brokerVersion attribute.
    Returns:
    The broker’s version number

getCCSID * §
    public int getCCSID()
    Get the character set of the queue manager.

getChannel * §
    public String getChannel()
    For client only, get the channel that was used.

getCleanupInterval * §
    public long getCleanupInterval()
    Get method for cleanupInterval attribute.
    Returns:
    How often the cleanup utility runs, in milliseconds

getCleanupLevel * §
    public int getCleanupLevel()
    Get method for cleanupLevel attribute.
    Returns:
    The value of cleanupLevel

getcldentId *
    public String getClientId()
    Get the client identifier that is used for all connections that are created using this ConnectionFactory.

description *
    public String getDescription()
    Retrieve the object description.
**ConnectionFactory**

getDirectAuth *

```java
public int getDirectAuth()
```

Get method for the direct authentication attribute.

**Returns:**

The value of the direct authentication attribute

**See also:**

setDirectAuth()

getFailIfQuiesce §

```java
public int getFailIfQuiesce()
```

Get the default behavior of applications accessing a quiescing queue manager when using destinations created using this ConnectionFactory object.

getHostName *

```java
public String getHostName()
```

Retrieve the name of the host.

getLocalAddress *

```java
public String getLocalAddress()
```

Get the local address.

**See also:**

setLocalAddress()

getMessageRetention *

```java
public int getMessageRetention()
```

Get method for messageRetention attribute.

**Returns:**

- JMSC.MQJMS_MRET_YES: unwanted messages remain on the input queue.
- JMSC.MQJMS_MRET_NO: unwanted messages are dealt with according to their disposition options.

getMessageSelection §

```java
public int getMessageSelection()
```

Get method for the message selection attribute.

**Returns:**

The value of the message selection attribute

**See also:**

setMessageSelection()
getMsgBatchSize * §

public int getMsgBatchSize()

Return the current value of this property.

getMulticast *

public int getMulticast()

Get method for the multicast attribute.

Returns:
An integer representing the current multicast setting.

See also:
setMulticast()

getPollingInterval * §

public int getPollingInterval()

Return the current value of this property.

getPort *

public int getPort()

For client connections or direct TCP/IP connection to WebSphere MQ Event Broker, get the port number.

getProxyHostName *

public String getProxyHostName()

Get method for the proxy host name attribute.

Returns:
The host name of the proxy server when establishing a direct connection, or null if no proxy server is used.

getProxyPort *

public int getProxyPort()

Get method for the proxy port attribute.

Returns:
The port number to connect to on the proxy server.

getPubAckInterval * §

public int getPubAckInterval()

Get method for pubAckInterval attribute.

Returns:
The interval, in number of messages, between publish requests that require acknowledgement from the broker.
ConnectionFactory

getQueueManager * §
  public String getQueueManager()

  Get the name of the queue manager.

getReceiveExit * §
  public String getReceiveExit()

  Get the name of the receive exit class.

getReceiveExitInit * §
  public String getReceiveExitInit()

  Get the initialization string that was passed to the receive exit class.

getAddress *
  public Reference getAddress() throws NamingException

  Return a reference for this connection factory.

  Returns:
  A reference for this object.

  Throws:
  NamingException.

getSecurityExit * §
  public String getSecurityExit()

  Get the name of the security exit class.

getSecurityExitInit * §
  public String getSecurityExitInit()

  Get the security exit initialization string.

getAddress * §
  public String getAddress()

  Get the name of the send exit class.

getAddressInit * §
  public String getAddressInit()

  Get the send exit initialization string.

getSparseSubscriptions *
  public boolean getSparseSubscriptions()

  Get method for the sparse subscriptions attribute.

  Returns:
  The value of the sparse subscriptions attribute

  See also:
    setSparseSubscriptions()
getSSLCertStores * §

public java.util.Collection getSSLCertStores()

Return a collection of CertStore objects. If setSSLCertStores() was used to set a collection of CertStore objects, the value returned from getSSLCertStores() is a copy of the original collection. If setSSLCertStores() was used to set a string detailing a list of LDAP URIs, this method returns a collection of CertStore objects representing the LDAP CRLs.

getSSLCertStoresAsString * §

public String getSSLCertStoresAsString()
    throws JMSException

Return the string of LDAP URIs, set with setSSLCertStores. Throws JMSException if a collection of CertStores was set.

getSSLCipherSuite * §

public String getSSLCipherSuite()

Return the CipherSuite used for SSL encryption.

getSSLPeerName * §

public String getSSLPeerName()

Return the distinguished name pattern used to validate the queue manager.

getSSLSocketFactory * §

public javax.net.ssl.SSLSocketFactory getSSLSocketFactory()

Return the SSLSocketFactory used with SSL encryption.

getStatusRefreshInterval * §

public int getStatusRefreshInterval()

Get method for statusRefreshInterval attribute.

Returns:
   The number of milliseconds between transactions to refresh publish/subscribe status.

getSubscriptionStore * §

public int getSubscriptionStore()

Get method for the SUBSTORE property.

Returns:
   An integer representing the current SUBSTORE property.
ConnectionFactory

getSyncpointAllGets * §
public boolean getSyncpointAllGets()

Return the current value of this property.

getTemporaryModel *
public String getTemporaryModel()

getTempQPrefix *
public String getTempQPrefix()

Get the prefix that is used to form the name of a WebSphere MQ dynamic queue.

Returns:
The prefix that is used to form the name of a WebSphere MQ dynamic queue.

getTransportType *
public int getTransportType()

Retrieve the transport type.

getUseConnectionPooling * §
public boolean getUseConnectionPooling()

Return the current value of this property.

setBrokerCCSubQueue * §
public void setBrokerCCSubQueue(String x) throws JMSException

Set method for brokerCCSubQueue attribute.

Parameters:
brokerSubQueue: the name of the nondurable subscription queue to use for a connection consumer.

setBrokerControlQueue * §
public void setBrokerControlQueue(String x) throws JMSException

Set method for brokerControlQueue attribute.

Parameters:
brokerControlQueue: the name of the broker control queue.

setBrokerPubQueue * §
public void setBrokerPubQueue(String x) throws JMSException

Set method for brokerPubQueue attribute.

Parameters:
brokerPubQueue: the name of the broker publish queue.
**ConnectionFactory**

```
setBrokerQueueManager * §
    public void setBrokerQueueManager(String x) throws JMSException

Set method for brokerQueueManager attribute.

Parameters:
    brokerQueueManager: the name of the broker’s queue manager.

setBrokerSubQueue * §
    public void setBrokerSubQueue(String x) throws JMSException

Set method for brokerSubQueue attribute.

Parameters:
    brokerSubQueue: the name of the nondurable subscription queue to use.

setBrokerVersion *
    public void setBrokerVersion(int x) throws JMSException

Set method for brokerVersion attribute.

Parameters:
    An integer representing one of the valid broker version number values. These are represented by the constants:
    JMSC.MQJMS_BROKER_V1
    JMSC.MQJMS_BROKER_V2

setCCSID * §
    public void setCCSID(int x) throws JMSException

Set the character set to be used when connecting to the queue manager. See [Table 13 on page 127](#) for a list of allowed values. We recommend that you use the default value (819) for most situations.

setChannel * §
    public void setChannel(String x) throws JMSException

For client only, set the channel to use.

setCleanupInterval * §
    public void setCleanupInterval(long x) throws JMSException

Set method for cleanupInterval attribute.

Parameters:
    How often the cleanup utility runs, in milliseconds

setCleanupLevel * §
    public void setCleanupLevel(int x) throws JMSException

Set method for cleanupLevel attribute.

Parameters:
    An integer representing one of the valid cleanup levels. These are represented by the constants:
    JMSC.MQJMS_CLEANUP_NONE
    JMSC.MQJMS_CLEANUP_SAFE
    JMSC.MQJMS_CLEANUP_STRONG
    JMSC.MQJMS_CLEANUP_AS_PROPERTY
```
ConnectionFactory

setClientId *
public void setClientId(String x)

Set the client Identifier to be used for all connections created using this connection.

--- WebSphere MQ Event Broker note ---
This method always throws an IllegalStateException when you make a direct connection to WebSphere MQ Event Broker.

setDescription *
public void setDescription(String x)

A short description of the object.

setDirectAuth *
public void setDirectAuth(int x) throws JMSException

Set method for the direct authentication attribute.

Parameters:
x: an integer specifying the type of direct authentication that is required. The following are symbolic constants that represent the valid values of the parameter:
  JMSC.MQJMS_DIRECTAUTH_BASIC
  JMSC.MQJMS_DIRECTAUTH_CERTIFICATE

setFailIfQuiesce *
public void setFailIfQuiesce(int fiqValue) throws JMSException

Set the default behavior of applications accessing a quiescing queue manager when using destinations created using this ConnectionFactory object.

Takes values of:
• JMSC.MQJMS_FIQ_YES (default)
• JMSC.MQJMS_FIQ_NO

setHostName *
public void setHostName(String hostname)

For client connections or direct TCP/IP connections to WebSphere MQ Event Broker, the name of the host to connect to.

setLocalAddress *
public void setLocalAddress(String localAddress) throws JMSException

Set the local address.

Parameters:
  localAddress: the local address to be used.

  The format of a local address is [ip-addr][low-port[,high-port]]. Here are some examples:
  9.20.4.98
  The channel binds to address 9.20.4.98 locally
The channel binds to address 9.20.4.98 locally and uses port 1000.

The channel binds to address 9.20.4.98 locally and uses a port in the range 1000 to 2000.

The channel binds to a port in the range 1000 to 2000 locally.

You can specify a host name instead of an IP address.

Specify a range of ports to allow for connections that are required internally as well as those explicitly used by an application. The number of ports required depends on the application and the facilities it uses. Typically, this is the number of sessions the application uses plus three or four additional ports. If an application is having difficulty making connections, increase the number of ports in the range.

Note that connection pooling has an effect on how quickly a port can be reused. In JMS, connection pooling is switched on by default and it might be some minutes before a port can be reused and connection errors may occur in the meantime.

For direct connections, the local address determines which of the local network interfaces is used for multicast connections. When specifying a local address for a direct connection, do not include a port number. A port number is not valid for multicast and, if specified, causes a failure at connect time.

Throws:
JMSException if the format of the local address is incorrect.

setMessageRetention *
public void setMessageRetention(int x) throws JMSException

Set method for messageRetention attribute.

Parameters:
Valid values are:
- JMSC.MQJMS_MRET_YES: unwanted messages remain on the input queue.
- JMSC.MQJMS_MRET_NO: unwanted messages are dealt with according to their disposition options. For more information on this, see "General principles for point-to-point messaging" on page 278.
**ConnectionFactory**

```
setMessageSelection * §
public void setMessageSelection(int x)

Set method for the message selection attribute.

Parameters:
  x: an integer indicating whether the client or the broker performs
  message selection. The following are symbolic constants that
  represent the valid values of the parameter:
    JMSC.MQJMS_MSEL_CLIENT
    JMSC.MQJMS_MSEL_BROKER

setMsgBatchSize * §
public void setMsgBatchSize(int x)

Set the maximum number of messages to be taken at once when using
asynchronous delivery.

setMulticast *
public void setMulticast(int x) throws JMSException

Set method for the multicast attribute.

Parameters:
  x: an integer specifying a multicast setting. The following are
  symbolic constants that represent the valid values of the parameter:
    JMSC.MQJMS_MULTICAST_DISABLED
    JMSC.MQJMS_MULTICAST_NOT_RELIABLE
    JMSC.MQJMS_MULTICAST_RELIABLE
    JMSC.MQJMS_MULTICAST_ENABLED

setPollingInterval * §
public void setPollingInterval(int x)

Set the interval between scans of all receivers during asynchronous
message delivery. The value is a number of milliseconds.

setPort *
public void setPort(int port) throws JMSException

Set the port for a client connection or direct TCP/IP connection to
WebSphere MQ Event Broker.

Parameters:
  port: the new value to use.

Throws:
  JMSException if the port is negative.
```
setProxyHostName *

public void setProxyHostName(String proxyHostName) throws JMSException

Set method for the proxy host name attribute.

Parameters:
proxyHostName: the host name of the proxy server when establishing a direct connection, or null if no proxy server is used.

setProxyPort *

public void setProxyPort(int proxyPort) throws JMSException

Set method for the proxy port attribute.

Parameters:
proxyPort: the port number of the proxy server when establishing a direct connection.

setPubAckInterval * §

public void setPubAckInterval(int x)

Set method for pubAckInterval attribute. The number of messages to publish between requiring acknowledgement from the broker. The default is 25. Applications do not normally alter this value, and must not rely on this acknowledgement.

Parameters:
pubAckInterval: the number of messages to use as an interval.

setQueueManager * §

public void setQueueManager(String x) throws JMSException

Set the name of the queue manager to connect to.

setReceiveExit *

public void setReceiveExit(String receiveExit)

The name of a class that implements a receive exit.

setReceiveExitInit *

public void setReceiveExitInit(String x)

Initialization string that is passed to the constructor of the receive exit class.

setSecurityExit *

public void setSecurityExit(String securityExit)

The name of a class that implements a security exit.

setSecurityExitInit *

public void setSecurityExitInit(String x)

Initialization string that is passed to the security exit constructor.
ConnectionFactory

```java
setSendExit * §
public void setSendExit(String sendExit)

The name of a class that implements a send exit.

setSendExitInit * §
public void setSendExitInit(String x)

Initialization string that is passed to the constructor of send exit.

setSparseSubscriptions *
public void setSparseSubscriptions(boolean x)

Set method for the sparse subscriptions attribute. A sparse subscription is
one that receives infrequent matching messages. The default value of this
attribute is false. A value of true might be required if an application using
sparse subscriptions fails to receive messages because of log overflow. If
you set the attribute to true, the application must be able to open the
consumer queue for browsing messages.

Parameters:
x: indicates whether sparse subscriptions are selected.

setSSLCertStores * §
public void setSSLCertStores(java.util.Collection stores)

Provide a collection of CertStore objects used for CRL checking. The
certificate provided by the queue manager is checked against one of the
CertStore objects contained within the collection; if the certificate is found,
the connection attempt fails. At connect-time, each CertStore in the
collection is tried in turn until one is successfully used to verify the queue
manager’s certificate. If set to null (the default), no checking of the queue
manager’s certificate is performed. This property is ignored if
sslCipherSuite is null. Use of this property requires Java 2 v1.4.

If CertStores are specified using this method, the MQConnectionFactory
cannot be bound into a JNDI namespace. Attempting to do so will result in
an exception.

Note: To use a CertStore successfully with a CRL hosted on an LDAP
server, make sure that your Java Software Development Kit (SDK) is
compatible with the CRL. Some SDKs require that the CRL conforms
to RFC 2587, which defines a schema for LDAP v2. Most LDAP v3
servers use RFC 2256 instead.

setSSLCertStores * §
public void setSSLCertStores(String storeSpec)
    throws JMSException

Specify a list of LDAP servers used for CRL checking. This string must
consist of a sequence of space-delimited LDAP URLs of the form
ldap://host[:port]. If no port is specified, the LDAP default of 389 is
assumed. The certificate provided by the queue manager is checked against
one of the listed LDAP CRL servers; if found, the connection fails. Each
LDAP server is tried in turn until one is successfully used to verify the
queue manager’s certificate. If set to null (the default), no checking of the
queue manager’s certificate is performed. Throws JMSException if the
supplied list of LDAP URIs is not valid. This property is ignored if sslCipherSuite is null. Use of this property requires Java 2 v1.4.

Note: To use a CertStore successfully with a CRL hosted on an LDAP server, make sure that your Java Software Development Kit (SDK) is compatible with the CRL. Some SDKs require that the CRL conforms to RFC 2587, which defines a schema for LDAP v2. Most LDAP v3 servers use RFC 2256 instead.

setSSLCipherSuite * §

public void setSSLCipherSuite(String cipherSuite)

Set this to the CipherSuite matching the CipherSpec set on the SVRCONN channel. If set to null (the default), no SSL encryption is performed. See Appendix H, “SSL CipherSuites supported by WebSphere MQ,” on page 487 for a list of CipherSuites and their associated CipherSpecs.

setSSLPeerName * §

public void setSSLPeerName(String peerName)

Sets sslPeerName to a distinguished name pattern. If sslCipherSuite is set, this variable can be used to ensure the correct queue manager is used. For a description of the format for this value, see “Using the distinguished name of the queue manager” on page 90. The distinguished name provided by the queue manager must match this pattern, or the connection attempt fails. If set to null (the default), no checking of the queue manager’s DN is performed. Throws JMSException if the supplied pattern is not valid. This property is ignored if sslCipherSuite is null.

setSSLSocketFactory * §

public void setSSLSocketFactory(javax.net.ssl.SSLSocketFactory sf)

Set the SSLSocketFactory for use with SSL encryption. Use this to customize all aspects of SSL encryption. For more information on constructing and customizing SSLSocketFactory instances, refer to your JSSE provider’s documentation. If set to null (default), the JSSE default SSLSocketFactory is used when SSL encryption is requested. This property is ignored if sslCipherSuite is null.

If a custom SSLSocketFactory is specified, the MQConnectionFactory cannot be bound into a JNDI namespace. Attempting to do so results in an exception.

setStatusRefreshInterval * §

public void setStatusRefreshInterval(int x)

Set method for statusRefreshInterval attribute.

Parameters:

statusRefreshInterval: the number of milliseconds between transactions to refresh publish/subscribe status.
ConnectionFactory

setSubscriptionStore * §
public void setSubscriptionStore(int x) throws JMSException

Set method for the SUBSTORE property.

Parameters:
SubStoreType: an integer representing one of the valid values of the
SUBSTORE property. The following symbolic constants represent
the valid values:
JMSC.MQJMS_SUBSTORE_QUEUE
JMSC.MQJMS_SUBSTORE_BROKER
JMSC.MQJMS_SUBSTORE_MIGRATE

setSyncpointAllGets * §
public void setSyncpointAllGets(boolean x)

Choose whether to do all GET operations within a syncpoint. The default
setting for this property is false. This allows GET operations not under
transaction management to perform more quickly.

setTemporaryModel *
public void setTemporaryModel(String x) throws JMSException

setTempQPrefix *
public void setTempQPrefix(java.lang.String tempQPrefix) throws JMSException

Set the prefix to be used to form the name of a WebSphere MQ dynamic
queue.

Parameters:
tempQPrefix: the prefix to be used to form the name of a
WebSphere MQ dynamic queue.

Throws:
JMSException if the string is null, empty, greater than 33 characters
in length, or consists solely of a single asterisk (*).

setTransportType *
publi c void setTransportType(int x) throws JMSException

Set the transport type to use. It can be one of the following:
JMSC.MQJMS_TP_BINDINGS_MQ
JMSC.MQJMS_TP_CLIENT_MQ_TCPIP
JMSC.MQJMS_TP_DIRECT_TCPIP
JMSC.MQJMS_TP_DIRECT_HTTP

setUseConnectionPooling * §
pUBLIC void setUseConnectionPooling(boolean x)

Choose whether to use connection pooling. If you set this to true, JMS
enables connection pooling for the lifetime of any connections created
through the ConnectionFactory. This also affects connections created with
UseConnectionPooling set to false; to disable connection pooling
throughout a JVM, ensure that all ConnectionFactories used within the
JVM have ConnectionPooling set to false. The default, and recommended,
value is true. You can disable connection pooling if, for example, your
applications run in an environment that performs its own pooling.
ConnectionMetaData

public interface ConnectionMetaData

WebSphere MQ class: MQConnectionMetaData

java.lang.Object
+----com.ibm.mq.jms.MQConnectionMetaData

ConnectionMetaData provides information that describes the connection.

WebSphere MQ constructor

MQConnectionMetaData

public MQConnectionMetaData()

Methods

getJMSMajorVersion

public int getJMSMajorVersion() throws JMSException

Get the JMS major version number.

Returns:
The JMS major version number.

Throws:
JMSException if an internal error occurs in JMS implementation during the metadata retrieval.

getJMSMinorVersion

public int getJMSMinorVersion() throws JMSException

Get the JMS minor version number.

Returns:
The JMS minor version number.

Throws:
JMSException if an internal error occurs in JMS implementation during the metadata retrieval.

getJMSProviderName

public java.lang.String getJMSProviderName() throws JMSException

Get the JMS provider name.

Returns:
The JMS provider name.

Throws:
JMSException if an internal error occurs in JMS implementation during the metadata retrieval.

getJMSVersion

public java.lang.String getJMSVersion() throws JMSException

Get the JMS version.
Returns:
The JMS version.

Throws:
JMSException if an internal error occurs in JMS implementation during the metadata retrieval.

getJMSXPropertyNames
public java.utilEnumeration getJMSXPropertyNames() throws JMSException

Get an enumeration of the names of the JMSX Properties supported by this connection.

Returns:
An enumeration of JMSX PropertyNames.

Throws:
JMSException if an internal error occurs in JMS implementation during the property names retrieval.

getProviderMajorVersion
public int getProviderMajorVersion() throws JMSException

Get the JMS provider major version number.

Returns:
The JMS provider major version number.

Throws:
JMSException - if an internal error occurs in JMS implementation during the metadata retrieval.

getProviderMinorVersion
public int getProviderMinorVersion() throws JMSException

Get the JMS provider minor version number.

Returns:
The JMS provider minor version number.

Throws:
JMSException if an internal error occurs in JMS implementation during the metadata retrieval.

getProviderVersion
public java.lang.String getProviderVersion() throws JMSException

Get the JMS provider version.

Returns:
The JMS provider version.

Throws:
JMSException if an internal error occurs in JMS implementation during the metadata retrieval.

toString *
public String toString()

Overrides:
toString in class Object.
DeliveryMode

public interface DeliveryMode

Delivery modes supported by JMS.

Fields

NON_PERSISTENT

public static final int NON_PERSISTENT

The lowest overhead delivery mode, because it does not require that the message be logged to stable storage.

PERSISTENT

public static final int PERSISTENT

Instruct the JMS provider to log the message to stable storage as part of the client’s send operation.
**Destination**

public interface **Destination**
Subinterfaces: **Queue**, **TemporaryQueue**, **TemporaryTopic**, and **Topic**

WebSphere MQ class: **MQDestination**

```
java.lang.Object
   +----com.ibm.mq.jms.MQDestination
```

The Destination object encapsulates provider-specific addresses.

See also: **Queue**, **TemporaryQueue**, **TemporaryTopic**, and **Topic**

**WebSphere MQ constructors**

**MQDestination**

    public MQDestination()

**Methods**

**getCCSID** *

    public int getCCSID()

    Get the name of the character set that is used by this destination.

**getDescription** *

    public StringgetDescription()

    Get the description of the object.

**getEncoding** *

    public int getEncoding()

    Get the encoding that is used for this destination.

**getExpiry** *

    public int getExpiry()

    Get the value of the expiry for this destination.

**getFailIfQuiesce** *

    public int getFailIfQuiesce()

    Get the behavior of applications accessing a quiescing queue manager with this destination.

**getPersistence** *

    public int getPersistence()

    Get the value of the persistence for this destination.

**getPriority** *

    public int getPriority()

    Get the override priority value.

**getTargetClient** *
public int getTargetClient()

Get the JMS compliance indicator flag.

setCCSID *

public void setCCSID(int x) throws JMSException

Character set to be used to encode text strings in messages sent to this destination. See Table 13 on page 127 for a list of allowed values. The default value is 1208 (UTF8).

setDescription *

public void setDescription(String x)

A short description of the object.

setEncoding *

public void setEncoding(int x) throws JMSException

The encoding to be used for numeric fields in messages sent to this destination. See Table 13 on page 127 for a list of allowed values.

setExpiry *

public void setExpiry(int expiry) throws JMSException

Override the expiry of all messages sent to this destination.

setFailIfQuiesce *

public void setFailIfQuiesce(int fiqValue) throws JMSException

Set the behavior of applications accessing a quiescing queue manager with this destination.

Takes values of:
- JMSC.MQJMS_FIQ_YES (default)
- JMSC.MQJMS_FIQ_NO

setPersistence *

public void setPersistence(int persistence) throws JMSException

Override the persistence of all messages sent to this destination.

setPriority *

public void setPriority(int priority) throws JMSException

Override the priority of all messages sent to this destination.

setTargetClient *

public void setTargetClient(int targetClient) throws JMSException

Whether the remote application is JMS compliant.
ExceptionListener

public interface ExceptionListener

If a JMS provider 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 that describes the problem.

This allows a client to be asynchronously notified of a problem. Some connections only consume messages, so they have no other way to learn that their Connection has failed.

Exceptions are delivered when:
• There is a failure in receiving an asynchronous message
• A message throws a runtime exception

Methods

onException

public void onException(JMSException exception)

Notify user of a JMS exception.

Parameters:

exception: the JMS exception. These are exceptions that result from asynchronous message delivery. Typically, they indicate a problem with receiving a message from the queue manager, or possibly an internal error in the JMS implementation.
MapMessage

public interface MapMessage
extends Message

WebSphere MQ class: JMSMapMessage

java.lang.Object
  +----com.ibm.jms.JMSMessage
      +----com.ibm.jms.JMSMapMessage

Use a MapMessage to send a set of name-value pairs where names are strings and values are Java primitive types. The entries can be accessed sequentially or randomly by name. The order of the entries is undefined.

See also: BytesMessage, Message, ObjectMessage, StreamMessage, and TextMessage

Methods

getBoolean

public boolean getBoolean(java.lang.String name) throws JMSException

Return the boolean value with the given name.

Parameters:
  name: the name of the boolean

Returns:
  The boolean value with the given name.

Throws:
  • JMSException if JMS fails to read the message because of an internal JMS error.
  • MessageFormatException if this type conversion is not valid.

getByte

public byte getByte(java.lang.String name) throws JMSException

Return the byte value with the given name.

Parameters:
  name: the name of the byte.

Returns:
  The byte value with the given name.

Throws:
  • JMSException if JMS fails to read the message because of an internal JMS error.
  • MessageFormatException if this type conversion is not valid.
MapMessage

getBytes

```java
public byte[] getBytes(java.lang.String name) throws JMSException
```

Return the byte array value with the given name.

**Parameters:**
- `name`: the name of the byte array.

**Returns:**
A copy of the byte array value with the given name. If there is no item by this name, a null value is returned.

**Throws:**
- JMSException if JMS fails to read the message because of an internal JMS error.
- MessageFormatException if this type of conversion is not valid.

getChar

```java
public char getChar(java.lang.String name) throws JMSException
```

Return the Unicode character value with the given name.

**Parameters:**
- `name`: the name of the Unicode character.

**Returns:**
The Unicode character value with the given name.

**Throws:**
- JMSException if JMS fails to read the message because of an internal JMS error.
- MessageFormatException if this type conversion is not valid.

getDouble

```java
public double getDouble(java.lang.String name) throws JMSException
```

Return the double value with the given name.

**Parameters:**
- `name`: the name of the double.

**Returns:**
The double value with the given name.

**Throws:**
- JMSException if JMS fails to read the message because of an internal JMS error.
- MessageFormatException if this type conversion is not valid.
getFloat
public float getFloat(java.lang.String name) throws JMSException

Return the float value with the given name.

Parameters:
   name: the name of the float.

Returns:
The float value with the given name.

Throws:
   • JMSException if JMS fails to read the message because of an internal JMS error.
   • MessageFormatException if this type conversion is not valid.

getInt
public int getInt(java.lang.String name) throws JMSException

Return the integer value with the given name.

Parameters:
   name: the name of the integer.

Returns:
The integer value with the given name.

Throws:
   • JMSException if JMS fails to read the message because of an internal JMS error.
   • MessageFormatException if this type conversion is not valid.

getLong
public long getLong(java.lang.String name) throws JMSException

Return the long value with the given name.

Parameters:
   name: the name of the long.

Returns:
The long value with the given name.

Throws:
   • JMSException if JMS fails to read the message because of an internal JMS error.
   • MessageFormatException if this type conversion is not valid.

getMapNames
public java.util Enumeration getMapNames() throws JMSException

Return an enumeration of all the map message’s names.

Returns:
An enumeration of all the names in this map message.

Throws:
JMSException if JMS fails to read the message because of an internal JMS error.
MapMessage

**getObject**

```java
public java.lang.Object getObject(java.lang.String name) throws JMSException
```

Return the Java object value with the given name. This method returns in object format, a value that has been stored in the map either using the setObject method call, or the equivalent primitive set method.

**Parameters:**
- name: the name of the Java object.

**Returns:**
- A copy of the Java object value with the given name, in object format (if it is set as an int, an Integer is returned). If there is no item by this name, a null value is returned.

**Throws:**
- JMSException if JMS fails to read the message because of an internal JMS error.

**getShort**

```java
public short getShort(java.lang.String name) throws JMSException
```

Return the short value with the given name.

**Parameters:**
- name: the name of the short.

**Returns:**
- The short value with the given name.

**Throws:**
- JMSException if JMS fails to read the message because of an internal JMS error.
- MessageFormatException if this type conversion is not valid.

**getString**

```java
public java.lang.String getString(java.lang.String name) throws JMSException
```

Return the string value with the given name.

**Parameters:**
- name: the name of the string.

**Returns:**
- The string value with the given name. If there is no item by this name, a null value is returned.

**Throws:**
- JMSException if JMS fails to read the message because of an internal JMS error.
- MessageFormatException if this type conversion is not valid.
itemExists

public boolean itemExists(java.lang.String name)
    throws JMSException

Check if an item exists in this MapMessage.

Parameters:
    name: the name of the item to test.

Returns:
    True if the item exists.

Throws:
    JMSException - if a JMS error occurs.

setBoolean

public void setBoolean(java.lang.String name,
            boolean value) throws JMSException

Set a boolean value with the given name into the map.

Parameters:
    • name: the name of the boolean.
    • value: the boolean value to set in the Map.

Throws:
    • JMSException if JMS fails to write message due to some internal JMS error.
    • MessageNotWriteableException if the message is in read-only mode.

setByte

public void setByte(java.lang.String name,
            byte value) throws JMSException

Set a byte value with the given name into the map.

Parameters:
    • name: the name of the byte.
    • value: the byte value to set in the Map.

Throws:
    • JMSException if JMS fails to write message due to some internal JMS error.
    • MessageNotWriteableException if the message is in read-only mode.

setBytes

public void setBytes(java.lang.String name,
            byte[] value) throws JMSException

Set a byte array value with the given 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 the value in the map is not altered by subsequent modifications to the array.
MapMessage

Throws:

- JMSException if JMS fails to write message due to some internal JMS error.
- MessageNotWritableException if the message is in read-only mode.

setBytes

public void setBytes(java.lang.String name, byte[] value, int offset, int length) throws JMSException

Set a portion of the byte array value with the given name into the mp.

The array is copied, so the value in the map is not altered by subsequent modifications to the array.

Parameters:

- name: the name of the byte array.
- value: the byte array value to set in the Map.
- offset: the initial offset within the byte array.
- length: the number of bytes to be copied.

Throws:

- JMSException if JMS fails to write message due to some internal JMS error.
- MessageNotWritableException if the message is in read-only mode.

setChar

public void setChar(java.lang.String name, char value) throws JMSException

Set a Unicode character value with the given name into the map.

Parameters:

- name: the name of the Unicode character.
- value: the Unicode character value to set in the map.

Throws:

- JMSException if JMS fails to write message due to some internal JMS error.
- MessageNotWritableException if the message is in read-only mode.

setDouble

public void setDouble(java.lang.String name, double value) throws JMSException

Set a double value with the given name into the map.

Parameters:

- name: the name of the double.
- value: the double value to set in the Map.

Throws:
MapMessage

- JMSException if JMS fails to write message due to some internal JMS error.
- MessageNotWriteableException if the message is in read-only mode.

**setFloat**

```java
public void setFloat(String name, float value) throws JMSException
```

Set a float value with the given name into the map.

**Parameters:**
- name: the name of the float.
- value: the float value to set in the map.

**Throws:**
- JMSException if JMS fails to write message due to some internal JMS error.
- MessageNotWriteableException if the message is in read-only mode.

**setInt**

```java
public void setInt(String name, int value) throws JMSException
```

Set an integer value with the given name into the map.

**Parameters:**
- name: the name of the integer.
- value: the integer value to set in the map.

**Throws:**
- JMSException if JMS fails to write message due to some internal JMS error.
- MessageNotWriteableException if the message is in read-only mode.

**setLong**

```java
public void setLong(String name, long value) throws JMSException
```

Set a long value with the given name into the map.

**Parameters:**
- name: the name of the long.
- value: the long value to set in the map.

**Throws:**
- JMSException if JMS fails to write message due to some internal JMS error.
- MessageNotWriteableException if the message is in read-only mode.
MapMessage

setObject

```java
public void setObject(java.lang.String name,
                      java.lang.Object value) throws JMSException
```

Set a Java object value with the given name into the map. This method works only for object primitive types (for example, Integer, Double, and Long), strings and byte arrays.

**Parameters:**

- **name**: the name of the Java object.
- **value**: the Java object value to set in the map.

**Throws:**

- JMSException if JMS fails to write message due to some internal JMS error.
- MessageFormatException if object is not valid.
- MessageNotWriteableException if the message is in read-only mode.

setShort

```java
public void setShort(java.lang.String name,
                     short value) throws JMSException
```

Set a short value with the given name into the map.

**Parameters:**

- **name**: the name of the short.
- **value**: the short value to set in the map.

**Throws:**

- JMSException if JMS fails to write message due to some internal JMS error.
- MessageNotWriteableException - if the message is in read-only mode.

setString

```java
public void setString(java.lang.String name,
                      java.lang.String value) throws JMSException
```

Set a string value with the given name into the map.

**Parameters:**

- **name**: the name of the string.
- **value**: the string value to set in the map.

**Throws:**

- JMSException if JMS fails to write message due to some internal JMS error.
- MessageNotWriteableException if the message is in read-only mode.
Message

public interface Message

WebSphere MQ class: JMSMessage

java.lang.Object
   com.ibm.jms.MQJMSMessage

The Message interface is the root interface of all JMS messages. It defines the JMS header and the acknowledge method used for all messages.

Fields

DEFAULT_DELIVERY_MODE
   public static final int DEFAULT_DELIVERY_MODE

The default delivery mode value.

DEFAULT_PRIORITY
   public static final int DEFAULT_PRIORITY

The default priority value.

DEFAULT_TIME_TO_LIVE
   public static final long DEFAULT_TIME_TO_LIVE

The default time-to-live value.

Methods

acknowledge
   public void acknowledge() throws JMSException

Acknowledge this and all previous messages received by the session.

Throws:
   JMSException if JMS fails to acknowledge because of an internal JMS error.

clearBody
   public void clearBody() throws JMSException

Clear out the message body. All other parts of the message are left untouched.

Throws:
   JMSException if JMS fails to because of an internal JMS error.
clearProperties

clearProperties()

Clear a message’s properties. The header fields and message body are not cleared.

Throws:
JMSException if JMS fails to clear JMS message properties because of an internal JMS error.

getBooleanProperty

getBooleanProperty(String name)

Return the boolean property value with the given name.

Parameters:
name: the name of the boolean property.

Returns:
The boolean property value with the given name.

Throws:
JMSException if JMS fails to get the property because of an internal JMS error.
MessageFormatException if this type conversion is not valid.

getByteProperty

getByteProperty(String name)

Return the byte property value with the given name.

Parameters:
name: the name of the byte property.

Returns:
The byte property value with the given name.

Throws:
JMSException if JMS fails to get the property because of an internal JMS error.
MessageFormatException if this type conversion is not valid.

getDoubleProperty

g(double name)

Return the double property value with the given name.

Parameters:
name: the name of the double property.

Returns:
The double property value with the given name.

Throws:
JMSException if JMS fails to get the property because of an internal JMS error.
MessageFormatException if this type conversion is not valid.
**getMessage**

```java
public byte[] getMessage() throws JMSException
```

Get the message as a byte array.

**Parameters:**

name: the name of the float property.

**Returns:**

The float property value with the given name.

**Throws:**

- JMSException if JMS fails to get the property because of an internal JMS error.
- MessageFormatException if this type conversion is not valid.

---

**getFloatProperty**

```java
public float getFloatProperty(java.lang.String name) throws JMSException
```

Return the float property value with the given name.

**Parameters:**

name: the name of the float property.

**Returns:**

The float property value with the given name.

**Throws:**

- JMSException if JMS fails to get the property because of an internal JMS error.
- MessageFormatException if this type conversion is not valid.

---

**getIntProperty**

```java
public int getIntProperty(java.lang.String name) throws JMSException
```

Return the integer property value with the given name.

**Parameters:**

name: the name of the integer property.

**Returns:**

The integer property value with the given name.

**Throws:**

- JMSException if JMS fails to get the property because of an internal JMS error.
- MessageFormatException if this type conversion is not valid.

---

**getJMSCorrelationID**

```java
public java.lang.String getJMSCorrelationID() throws JMSException
```

Get the correlation ID for the message.

**Returns:**

The correlation ID of a message as a string.

**Throws:**

JMSException if JMS fails to get the correlation ID because of an internal JMS error.

**See also:**

setJMSCorrelationID(), getJMSCorrelationIDAsBytes(), setJMSCorrelationIDAsBytes()
**getJMSCorrelationIDAsBytes**

```java
public byte[] getJMSCorrelationIDAsBytes() throws JMSException
```

Get the correlation ID as an array of bytes for the message.

**Returns:**

The correlation ID of a message as an array of bytes.

**Throws:**

JMSException if JMS fails to get correlation ID because of an internal JMS error.

**See also:**

setJMSCorrelationID(), getJMSCorrelationID(), setJMSCorrelationIDAsBytes()

**getJMSDeliveryMode**

```java
public int getJMSDeliveryMode() throws JMSException
```

Get the delivery mode for this message.

**Returns:**

The delivery mode of this message.

**Throws:**

JMSException if JMS fails to get JMS DeliveryMode because of an internal JMS error.

**See also:**

setJMSDeliveryMode(), DeliveryMode

**getJMSDestination**

```java
public Destination getJMSDestination() throws JMSException
```

Get the destination for this message.

**Returns:**

The destination of this message.

**Throws:**

JMSException if JMS fails to get JMS Destination because of an internal JMS error.

**See also:**

setJMSDestination()

**getJMSExpiration**

```java
public long getJMSExpiration() throws JMSException
```

Get the message’s expiration value.

**Returns:**

The time that the message expires. It is the sum of the time-to-live value specified by the client, and the GMT at the time of the send.

**Throws:**

JMSException if JMS fails to get JMS message expiration because of an internal JMS error.

**See also:**

setJMSExpiration()
getJMSMessageID

```java
public java.lang.String getJMSMessageID() throws JMSException
```

Get the message ID.

**Returns:**
The message ID.

**Throws:**
JMSException if JMS fails to get the message ID because of an internal JMS error.

**See also:**
setJMSMessageID()

getJMSPriority

```java
public int getJMSPriority() throws JMSException
```

Get the message priority.

**Returns:**
The message priority.

**Throws:**
JMSException if JMS fails to get JMS message priority because of an internal JMS error.

**See also:**
setJMSPriority() for priority levels

getJMSRedelivered

```java
public boolean getJMSRedelivered() throws JMSException
```

Get an indication of whether this message is being redelivered.

If a client receives a message with the redelivered indicator set, it is likely, but not guaranteed, that this message was delivered to the client earlier, but that the client did not acknowledge its receipt at that earlier time.

**Returns:**
Set to true if this message is being redelivered.

**Throws:**
JMSException if JMS fails to get JMS redelivered flag because of an internal JMS error.

**See also:**
setJMSRedelivered()

getJMSReplyTo

```java
public Destination getJMSReplyTo() throws JMSException
```

Get where a reply to this message should be sent.

**Returns:**
Where to send a response to this message

**Throws:**
JMSException if JMS fails to get ReplyTo destination because of an internal JMS error.
See also: 
   setJMSReplyTo()

**getJMSTimestamp**

public long **getJMSTimestamp**() throws JMSException

Get the message timestamp.

**Returns:**
   The message timestamp.

**Throws:**
   JMSException if JMS fails to get the timestamp because of an internal JMS error.

See also: 
   setJMSTimestamp()

**getJMSType**

public java.lang.String **getJMSType**() throws JMSException

Get the message type.

**Returns:**
   The message type.

**Throws:**
   JMSException if JMS fails to get JMS message type because of an internal JMS error.

See also: 
   setJMSType()

**getLongProperty**

public long **getLongProperty**(java.lang.String name) throws JMSException

Return the long property value with the given name.

**Parameters:**
   name: the name of the long property.

**Returns:**
   The long property value with the given name.

**Throws:**
   - JMSException if JMS fails to get the property because of an internal JMS error.
   - MessageFormatException if this type conversion is not valid.

**getObjectProperty**

public java.lang.Object **getObjectProperty**(java.lang.String name) throws JMSException

Return the Java object property value with the given name.

**Parameters:**
   name: the name of the Java object property.

**Returns:**
   The Java object property value with the given name, in object format (for example, if it set as an int, an Integer is returned). If there is no property by this name, a null value is returned.
Throws:
JMSException if JMS fails to get the property because of an internal JMS error.

getPropertyNames
public java.util.Enumeration getPropertyName() throws JMSException

Return an enumeration of all the property names.

Returns:
An enumeration of all the names of property values.

Throws:
JMSException if JMS fails to get the property names because of an internal JMS error.

getShortProperty
public short getShortProperty(java.lang.String name) throws JMSException

Return the short property value with the given name.

Parameters:
name: the name of the short property.

Returns:
The short property value with the given name.

Throws:
• JMSException if JMS fails to get the property because of an internal JMS error.
• MessageFormatException if this type conversion is not valid.

getStringProperty
public java.lang.String getStringProperty(java.lang.String name) throws JMSException

Return the string property value with the given name.

Parameters:
name: the name of the string property

Returns:
The string property value with the given name. If there is no property by this name, a null value is returned.

Throws:
• JMSException if JMS fails to get the property because of an internal JMS error.
• MessageFormatException if this type conversion is not valid.
propertyExists
   public boolean propertyExists(java.lang.String name)
   throws JMSException

Check if a property value exists.

Parameters:
   name: the name of the property to test.

Returns:
   True if the property does exist.

Throws:
   JMSException if JMS fails to check whether a property exists
   because of an internal JMS error.

setBooleanProperty
   public void setBooleanProperty(java.lang.String name,
       boolean value) throws JMSException

Set a boolean property value with the given name into the message.

Parameters:
   • name: the name of the boolean property.
   • value: the boolean property value to set in the message.

Throws:
   • JMSException if JMS fails to set property because of an internal
     JMS error.
   • MessageNotWriteableException if the properties are read-only.

setByteProperty
   public void setByteProperty(java.lang.String name,
       byte value) throws JMSException

Set a byte property value with the given name into the message.

Parameters:
   • name: the name of the byte property.
   • value: the byte property value to set in the message.

Throws:
   • JMSException if JMS fails to set property because of an internal
     JMS error.
   • MessageNotWriteableException if the properties are read-only.

setDoubleProperty
   public void setDoubleProperty(java.lang.String name,
       double value) throws JMSException

Set a double property value with the given name into the message.

Parameters:
   • name: the name of the double property.
   • value: the double property value to set in the message.
Throws:
- JMSException if JMS fails to set the property because of an internal JMS error.
- MessageNotWriteableException if the properties are read-only.

**setFloatProperty**
public void setFloatProperty(java.lang.String name, float value) throws JMSException

Set a float property value with the given name into the message.

**Parameters:**
- name: the name of the float property.
- value: the float property value to set in the message.

**Throws:**
- JMSException if JMS fails to set the property because of an internal JMS error.
- MessageNotWriteableException if the properties are read-only.

**setIntProperty**
public void setIntProperty(java.lang.String name, int value) throws JMSException

Set an integer property value with the given name into the message.

**Parameters:**
- name: the name of the integer property.
- value: the integer property value to set in the message.

**Throws:**
- JMSException if JMS fails to set the property because of an internal JMS error.
- MessageNotWriteableException if the properties are read-only.

**setJMSCorrelationID**
public void setJMSCorrelationID(java.lang.String correlationID)
throws JMSException

Set 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.

**Note:** The use of a byte[] value for JMSCorrelationID is non-portable.

**Parameters:**
- correlationID: the message ID of a message being referred to.

**Throws:**
- JMSException if JMS fails to set the correlation ID because of an internal JMS error.

**See also:**
- getJMSCorrelationID(), getJMSCorrelationIDAsBytes(), setJMSCorrelationIDAsBytes()
setJMSCorrelationIDAsBytes
public void setJMSCorrelationIDAsBytes(byte[] correlationID)
throws JMSException

Set the correlation ID as an array of bytes for the message. A client can use this call to set the correlationID equal either to a messageID from a previous message, or to an application-specific string. Application-specific strings must not start with the characters ID.

Parameters:
- correlationID: the correlation ID as a string, or the message ID of a message being referred to.

Throws:
- JMSException if JMS fails to set the correlation ID because of an internal JMS error.

See also:
- setJMSCorrelationID(), getJMSCorrelationID(),
- getJMSCorrelationIDAsBytes()

setJMSDeliveryMode
public void setJMSDeliveryMode(int deliveryMode)
throws JMSException

Set the delivery mode for this message.

Any value set using this method is ignored when the message is sent, but this method can be used to change the value in a received message.

To alter the delivery mode when a message is sent, use the setDeliveryMode method on the QueueSender or TopicPublisher (this method is inherited from MessageProducer).

Parameters:
- deliveryMode: the delivery mode for this message.

Throws:
- JMSException if JMS fails to set JMS DeliveryMode because of an internal JMS error.

See also:
- getJMSDeliveryMode(), DeliveryMode

setJMSDestination
public void setJMSDestination(Destination destination)
throws JMSException

Set the destination for this message.

Any value set using this method is ignored when the message is sent, but this method can be used to change the value in a received message.

Parameters:
- destination: the destination for this message.

Throws:
- JMSException if JMS fails to set JMS destination because of an internal JMS error.
See also:
   getJMSDestination()

setJMSExpiration
   public void setJMSExpiration(long expiration) throws JMSException

Set the message’s expiration value.

Any value set using this method is ignored when the message is sent, but this method can be used to change the value in a received message.

Parameters:
   expiration: the message’s expiration time.

Throws:
   JMSException if JMS fails to set JMS message expiration because of an internal JMS error.

See also:
   getJMSExpiration()

setJMSMessageID
   public void setJMSMessageID(java.lang.String id) throws JMSException

Set the message ID.

Any value set using this method is ignored when the message is sent, but this method can be used to change the value in a received message.

Parameters:
   id: the ID of the message.

Throws:
   JMSException if JMS fails to set the message ID because of an internal JMS error.

See also:
   getJMSMessageID()

setJMSPriority
   public void setJMSPriority(int priority) throws JMSException

Set the priority for this message.

JMS defines a ten-level priority value, with 0 as the lowest priority, and 9 as the highest. In addition, clients must consider priorities 0-4 as gradations of normal priority, and priorities 5-9 as gradations of expedited priority.

Parameters:
   priority: the priority of this message.

Throws:
   JMSException if JMS fails to set JMS message priority because of an internal JMS error.

See also:
   getJMSPriority()
setJMSRedelivered

```java
public void setJMSRedelivered(boolean redelivered)
    throws JMSException
```

Set to indicate whether this message is being redelivered.

Any value set using this method is ignored when the message is sent, but this method can be used to change the value in a received message.

**Parameters:**
- `redelivered`: an indication of whether this message is being redelivered.

**Throws:**
- `JMSException` if JMS fails to set JMSRedelivered flag because of an internal JMS error.

**See also:**
- `getJMSRedelivered()`

setJMSReplyTo

```java
public void setJMSReplyTo(Destination replyTo)
    throws JMSException
```

Set where a reply to this message should be sent.

**Parameters:**
- `replyTo`: where to send a response to this message. A null value indicates that no reply is expected.

**Throws:**
- `JMSException` if JMS fails to set ReplyTo destination because of an internal JMS error.

**See also:**
- `getJMSReplyTo()`

setJMSTimestamp

```java
public void setJMSTimestamp(long timestamp)
    throws JMSException
```

Set the message timestamp.

Any value set using this method is ignored when the message is sent, but this method can be used to change the value in a received message.

**Parameters:**
- `timestamp`: the timestamp for this message.

**Throws:**
- `JMSException` if JMS fails to set the timestamp because of an internal JMS error.

**See also:**
- `getJMSTimestamp()`
setJMSType

public void setJMSType(java.lang.String type) throws JMSException

Set the message type.

JMS clients must assign a value to type whether the application makes use of it or not. This ensures that it is properly set for those providers that require it.

Parameters:
- type: the class of message.

Throws:
- JMSException if JMS fails to set JMS message type because of an internal JMS error.

See also:
- getJMSType()

setLongProperty

public void setLongProperty(java.lang.String name, long value) throws JMSException

Set a long property value with the given name into the message.

Parameters:
- name: the name of the long property.
- value: the long property value to set in the message.

Throws:
- JMSException if JMS fails to set property because of an internal JMS error.
- MessageNotWriteableException if the properties are read-only.

setObjectProperty

public void setObjectProperty(java.lang.String name, java.lang.Object value) throws JMSException

Set a property value with the given name into the message.

Parameters:
- name: the name of the Java object property.
- value: the Java object property value to set in the message.

Throws:
- JMSException if JMS fails to set property because of an internal JMS error.
- MessageFormatException if the object is not valid.
- MessageNotWriteableException - if the properties are read-only.
**Message**

`setShortProperty`

```java
public void setShortProperty(java.lang.String name,
                               short value) throws JMSException
```

Set a short property value with the given name into the message.

**Parameters:**

- `name`: the name of the short property.
- `value`: the short property value to set in the message.

**Throws:**

- `JMSException` if JMS fails to set property because of an internal JMS error.
- `MessageNotWriteableException` if the properties are read-only.

`setStringProperty`

```java
public void setStringProperty(java.lang.String name,
                               java.lang.String value) throws JMSException
```

Set a string property value with the given name into the message.

**Parameters:**

- `name`: the name of the string property.
- `value`: the string property value to set in the message.

**Throws:**

- `JMSException` if JMS fails to set the property because of an internal JMS error.
- `MessageNotWriteableException` if the properties are read-only.
**MessageConsumer**

public interface **MessageConsumer**

Subinterfaces: **QueueReceiver** and **TopicSubscriber**

WebSphere MQ class: **MQMessageConsumer**

java.lang.Object
   +----com.ibm.mq.jms.MQMessageConsumer

MessageConsumer is the parent interface for all message consumers. A client uses a message consumer to receive messages from a Destination.

See also: **QueueReceiver**, **Session**, and **TopicSubscriber**

**Methods**

close

    public void close() throws JMSException

Close the message consumer.

Because a provider can allocate some resources outside the JVM on behalf of a MessageConsumer, clients must close them when they are not needed. You cannot rely on garbage collection to reclaim these resources eventually, because this might not occur soon enough.

This call blocks until a receive or message listener in progress has completed.

**Throws:**

    JMSException if JMS fails to close the consumer because of an error.

getMessageListener

    public MessageListener getMessageListener() throws JMSException

Get the message consumer’s MessageListener.

**Returns:**

    The listener for the message consumer, or null if a listener is not set.

**Throws:**

    JMSException if JMS fails to get the message listener because of a JMS error.

See also:

    setMessageListener
getMessageSelector

public java.lang.String getMessageSelector() throws JMSException

Get this message consumer’s message selector expression.

Returns:
The message consumer’s message selector.

Throws:
JMSException if JMS fails to get the message selector because of a JMS error.

receive

public Message receive() throws JMSException

Receive the next message produced for this message consumer.

Returns:
The next message produced for this message consumer.

Throws:
JMSException if JMS fails to receive the next message because of an error.

receive

public Message receive(long timeout) throws JMSException

Receive the next message that arrives within the specified timeout interval. A timeout value of zero causes the call to wait indefinitely until a message arrives.

Parameters:
timeout: the timeout value (in milliseconds).

Returns:
The next message produced for this message consumer, or null if one is not available.

Throws:
JMSException if JMS fails to receive the next message because of an error.

receiveNoWait

public Message receiveNoWait() throws JMSException

Receive the next message if one is immediately available.

Returns:
The next message produced for this message consumer, or null if one is not available.

Throws:
JMSException if JMS fails to receive the next message because of an error.
setMessageListener

public void setMessageListener(MessageListener listener)
throws JMSException

Set the message consumer’s MessageListener.

Parameters:
messageListener: the messages are delivered to this listener.

Throws:
JMSException if JMS fails to set message listener because of a JMS error.

See also:
getMessageListener
public interface MessageListener

Use a MessageListener to receive asynchronously delivered messages.

Methods

onMessage

public void onMessage(Message message)

Pass a message to the listener.

Parameters:
message: the message passed to the listener.

See also
Session.setMessageListener
MessageProducer

public interface MessageProducer
Subinterfaces: QueueSender and TopicPublisher

WebSphere MQ class: MQMessageProducer

java.lang.Object
  +---com.ibm.mq.jms.MQMessageProducer

A client uses a MessageProducer to send messages to a destination.

See also: QueueSender, TopicPublisher, and Session.createProducer(javax.jms.Destination)

WebSphere MQ constructors

MQMessageProducer
  public MQMessageProducer()

Methods

close
  public void close() throws JMSException

Because a provider can allocate some resources outside the JVM on behalf
of a MessageProducer, clients must close them when they are not needed.
You cannot rely on garbage collection to reclaim these resources eventually,
because this might not occur soon enough.

Throws:
  JMSException if JMS fails to close the producer because of an error.

getDeliveryMode
  public int getDeliveryMode() throws JMSException

Get the producer’s default delivery mode.

Returns:
  The message delivery mode for this message producer.

Throws:
  JMSException if JMS fails to get the delivery mode because of an
  internal error.

See also:
  setDeliveryMode
getMessageProducer

getDestination (JMS 1.1 only)

public Destination getDestination() throws JMSException

Get the destination associated with the message producer.

Returns:

   The message producer’s destination.

Throws:

   JMSException if JMS fails to get the destination because of an internal JMS error.

getDisableMessageID

public boolean getDisableMessageID() throws JMSException

Get an indication of whether message IDs are disabled.

Returns:

   An indication of whether message IDs are disabled.

Throws:

   JMSException if JMS fails to get the disabled message ID because of an internal error.

getDisableMessageTimestamp

public boolean getDisableMessageTimestamp() throws JMSException

Get an indication of whether message timestamps are disabled.

Returns:

   An indication of whether message IDs are disabled.

Throws:

   JMSException if JMS fails to get the disabled message timestamp because of an internal error.

getPriority

public int getPriority() throws JMSException

Get the producer’s default priority.

Returns:

   The message priority for this message producer.

Throws:

   JMSException if JMS fails to get the priority because of an internal error.

See also:

   setPriority
getTimeToLive

public long getTimeToLive() throws JMSException

Get the default length of time in milliseconds from its dispatch time that the message system retains a produced message.

Returns:
The message time-to-live in milliseconds; zero is unlimited.

Throws:
JMSException if JMS fails to get the time-to-live because of an internal error.

See also:
setTimeToLive

send (JMS 1.1 only)

public void send(Message message) throws JMSException

Send a message using the message producer’s default delivery mode, default priority, and default time to live.

Parameters:
message: the message to send.

Throws:

- JMSException if JMS fails to send the message because of an internal JMS error.
- MessageFormatException if the message is not valid.
- InvalidDestinationException if a client uses this method with a message producer whose destination is not valid.
- java.lang.UnsupportedOperationException if a client uses this method with a message producer for which no destination was specified when it was created.

See also:
MessageProducer, Session.createProducer

send (JMS 1.1 only)

public void send(Message message, int deliveryMode, int priority, long timeToLive) throws JMSException

Send a message specifying a delivery mode, a priority, and a time to live.

Parameters:

- message: the message to send.
- deliveryMode: the delivery mode to use
- priority: the priority for the message
- timeToLive: the lifetime of the message in milliseconds.

Throws:

- JMSException if JMS fails to send the message because of an internal JMS error.
- MessageFormatException if the message is not valid.
- InvalidDestinationException if a client uses this method with a message producer whose destination is not valid.
• java.lang.UnsupportedOperationException if a client uses this
  method with a message producer for which no destination was
  specified when the message producer was created.

See also:
  Session.createProducer

send (JMS 1.1 only)

public void send(Destination destination, Message message)
  throws JMSException

Send a message to a destination if you are using a message producer for
which no destination was specified when the message producer was
created. The method uses the message producer’s default delivery mode,
default priority, and default time to live.

Typically, you specify a destination when you create a message producer
but, if you do not, you must specify a destination every time you send a
message.

Parameters:
  • destination: the destination to send the message to.
  • message: the message to send.

Throws:
  • JMSException if JMS fails to send the message because of an
    internal JMS error.
  • MessageFormatException if the message is not valid.
  • InvalidDestinationException if the destination is not valid.
  • java.lang.UnsupportedOperationException if a client uses this
    method with a message producer for which a destination was
    specified when the message producer was created.

See also:
  MessageProducer, Session.createProducer

send (JMS 1.1 only)

public void send(Destination destination, Message message,
  int deliveryMode, int priority,
  long timeToLive) throws JMSException

Send a message to a destination if you are using a message producer for
which no destination was specified when the message producer was
created. The method specifies a delivery mode, a priority, and a time to
live.

Typically, you specify a destination when you create a message producer
but, if do not, you must specify a destination every time you send a
message.

Parameters:
  • destination: the destination to which to send the message.
  • message: the message to send.
  • deliveryMode: the delivery mode to use
  • priority: the priority for the message
  • timeToLive: the lifetime of the message in milliseconds.
Throws:

- JMSException if JMS fails to send the message because of an internal JMS error.
- MessageFormatException if the message is not valid.
- InvalidDestinationException if the destination is not valid.
- java.lang.UnsupportedOperationException if a client uses this method with a message producer for which a destination was specified when the message producer was created.

See also:

Session.createProducer

setDeliveryMode

```java
class MessageProducer

public void setDeliveryMode(int deliveryMode) throws JMSException
```

Set the producer’s default delivery mode; it is set to DeliveryMode.PERSISTENT by default.

**Parameters:**
- deliveryMode: the message delivery mode for this message producer.

**Throws:**
- JMSException if JMS fails to set the delivery mode because of an internal error.

See also:
- getDeliveryMode, DeliveryMode.NON_PERSISTENT, DeliveryMode.PERSISTENT, Message.DEFAULT_DELIVERY_MODE

setDisableMessageID

```java
class MessageProducer

public void setDisableMessageID(boolean value) throws JMSException
```

Set whether message IDs are disabled; they are enabled by default.

**Note:** This method is ignored in the WebSphere MQ classes for Java Message Service implementation.

**Parameters:**
- value: indicates whether message IDs are disabled.

**Throws:**
- JMSException if JMS fails to set the disabled message ID because of an internal error.

setDisableMessageTimestamp

```java
class MessageProducer

public void setDisableMessageTimestamp(boolean value) throws JMSException
```

Set whether message timestamps are disabled; they are enabled by default.

**Note:** This method is ignored in the WebSphere MQ classes for Java Message Service implementation.

**Parameters:**
- value: indicates whether message timestamps are disabled.
MessageProducer

Throws:

JMSException if JMS fails to set the disabled message timestamp because of an internal error.

setPriority

public void setPriority(int defaultPriority) throws JMSException

Set the producer’s default priority (default 4).

Parameters:

defaultPriority: the message priority for this message producer.

Throws:

JMSException if JMS fails to set the priority because of an internal error.

See also:

getPriority, Message.DEFAULT_PRIORITY

setTimeToLive

public void setTimeToLive(long timeToLive) throws JMSException

Set the default length of time, in milliseconds from its dispatch time, that the message system retains a produced message.

Time-to-live is set to zero by default.

WebSphere MQ Event Broker note

This method throws a JMSException if set to other than 0 when you make a direct connection to WebSphere MQ Event Broker.

Parameters:

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

Throws:

JMSException if JMS fails to set the time-to-live because of an internal error.

See also:

g getTimeToLive, Message.DEFAULT_TIME_TO_LIVE
public class MQQueueEnumeration extends Object implements Enumeration

MQQueueEnumeration enumerates messages on a queue. This class is not defined in the JMS specification; it is created by calling the getEnumeration method of MQQueueBrowser. The class contains a base MQQueue instance to hold the browse cursor. The queue is closed once the cursor has moved off the end of the queue.

There is no way to reset an instance of this class; it acts as a *one-shot* mechanism.

See also: MQQueueBrowser

**Methods**

**hasMoreElements**

public boolean hasMoreElements()

Whether another message can be returned.

**nextElement**

public Object nextElement() throws NoSuchElementException

Return the current message.

If hasMoreElements() returns true, nextElement() always returns a message. It is possible for the returned message to pass its expiry date between the hasMoreElements() and the nextElement calls.
public interface ObjectMessage
extends Message

WebSphere MQ class: JMSObjectMessage

java.lang.Object
+----com.ibm.jms.JMSMessage
   +----com.ibm.jms.JMSObjectMessage

Use an ObjectMessage to send a message that contains a serializable Java object. It inherits from Message and adds a body containing a single Java reference. Only serializable Java objects can be used.

See also: BytesMessage, MapMessage, Message, StreamMessage, and TextMessage

Methods

getObject

public java.io.Serializable getObject()
     throws JMSException

Get the serializable object containing this message’s data. The default value is null.

Returns:
The serializable object containing this message’s data.

Throws:
• JMSException if JMS fails to get the object because of an internal JMS error.
• MessageFormatException if object deserialization fails.

setObject

public void setObject(java.io.Serializable object)
     throws JMSException

Set the serializable object containing this message’s data. The ObjectMessage contains a snapshot of the object at the time setObject() is called. Subsequent modifications of the object have no effect on the ObjectMessage body.

Parameters:
object: the message’s data.

Throws:
• JMSException if JMS fails to set the object because of an internal JMS error.
• MessageFormatException if object serialization fails.
• MessageNotWriteableException if the message is in read-only mode.
A Queue object encapsulates a provider-specific queue name. It is the way that a client specifies the identity of a queue to JMS methods.

**WebSphere MQ constructors**

MQQueue *

```java
public MQQueue()
```

Default constructor for use by the administration tool.

MQQueue *

```java
public MQQueue(String URIqueue)
```

Create a new MQQueue instance. The string takes a URI format, as described on page 204.

MQQueue *

```java
public MQQueue(String queueManagerName, String queueName)
```

**Methods**

**getBaseQueueManagerName** *

```java
public String getBaseQueueManagerName()
```

Returns:

The value of the WebSphere MQ queue manager name.

**getBaseQueueName** *

```java
public String getBaseQueueName()
```

Returns:

The value of the WebSphere MQ queue name.

**getQueueName**

```java
public java.lang.String getQueueName() throws JMSException
```

Get the name of this queue.

Clients that depend upon the name are not portable.

Returns:

The queue name
Queue

Throws:
JMSException if JMS implementation for queue fails to return the queue name because of an internal error.

gReference *

public Reference getReference() throws NamingException

Create a reference for this queue.

Returns:
A reference for this object.

Throws:
NamingException.

setBaseQueueManagerName *

public void setBaseQueueManagerName(String x) throws JMSException

Set the value of the WebSphere MQ queue manager name.

Note: Only the administration tool can use this method.

setBaseQueueName *

public void setBaseQueueName(String x) throws JMSException

Set the value of the WebSphere MQ queue name.

Note: Only the administration tool can use this method. It makes no attempt to decode queue:qmgr:queue format strings.

toString

public java.lang.String toString()

Return a well-formatted printed version of the queue name.

Returns:
The provider-specific identity values for this queue.

Overrides:
toString in class java.lang.Object
QueueBrowser

public interface QueueBrowser

WebSphere MQ class: MQQueueBrowser

java.lang.Object
   +----com.ibm.mq.jms.MQQueueBrowser

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

Note: The WebSphere MQ class MQQueueEnumeration is used to hold the browse cursor.

See also: QueueReceiver

Methods

close

public void close() throws JMSException

Because a provider can allocate some resources outside the JVM on behalf of a QueueBrowser, clients must close them when they are not needed. You cannot rely on garbage collection to reclaim these resources eventually, because this might not occur soon enough.

Throws:
   JMSException if a JMS fails to close this browser because of a JMS error.

getEnumeration

public java.utilEnumeration getEnumeration() throws JMSException

Get an enumeration for browsing the current queue messages in the order that they are received.

Returns:
   An enumeration for browsing the messages.

Throws:
   JMSException if JMS fails to get the enumeration for this browser because of a JMS error.

   Note: If the browser is created for a nonexistent queue, this is not detected until the first call to getEnumeration.

getMessageSelector

public java.lang.String getMessageSelector() throws JMSException

Get this queue browser’s message selector expression.

Returns:
   This queue browser’s message selector.

Throws:
   JMSException if JMS fails to get the message selector for this browser because of a JMS error.
QueueBrowser

getQueue

public Queue getQueue() throws JMSException

Get the queue associated with this queue browser.

Returns:
- The queue.

Throws:
- JMSException if JMS fails to get the queue associated with this browser because of a JMS error.
A QueueConnection is an active connection to a JMS point-to-point provider. A client uses a QueueConnection to create one or more QueueSessions for producing and consuming messages.

See also: Connection, QueueConnectionFactory, and XAQueueConnection

**Methods**

**close** *

```java
close()
```

public void close() throws JMSException

**Overrides:**

Close in class MQConnection.

**createConnectionConsumer**

```java
createConnectionConsumer()
```

public ConnectionConsumer createConnectionConsumer

```java
(Queue queue,
 java.lang.String messageSelector,
 ServerSessionPool sessionPool,
 int maxMessages)
```

throws JMSException

Create a connection consumer for this connection. This is an expert facility that is not used by regular JMS clients.

**Parameters:**

- queue: the queue to access.
- messageSelector: only messages with properties that match the message selector expression are delivered.
- 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.

**Throws:**

- JMSException if the JMS connection fails to create a connection consumer because of an internal error, or incorrect arguments for sessionPool and messageSelector.
- InvalidDestinationException if the queue is not valid.
- InvalidSelectorException if the message selector is not valid.
QueueConnection

See also:
  ConnectionConsumer

createQueueSession
  public QueueSession createQueueSession(boolean transacted,
                                          int acknowledgeMode)
                                                  throws JMSException

Create a QueueSession.

Parameters:
  - transacted: if true, the session is transacted.
  - acknowledgeMode: indicates whether the consumer or the client
                    acknowledges any messages it receives. Possible values are:
                      Session.AUTO_ACKNOWLEDGE
                      Session.CLIENT_ACKNOWLEDGE
                      Session.DUPS_OK_ACKNOWLEDGE

                    This parameter is ignored if the session is transacted.

Returns:
  A newly-created queue session.

Throws:
  JMSException if JMS Connection fails to create a session because of
                an internal error, or lack of support for specific transaction and
                acknowledgement mode.
QueueConnectionFactory

public interface QueueConnectionFactory
extends ConnectionFactory
Subinterfaces: XAQueueConnectionFactory

WebSphere MQ class: MQQueueConnectionFactory

java.lang.Object
  +----com.ibm.mq.jms.MQConnectionFactory
  |    +----com.ibm.mq.jms.MQQueueConnectionFactory

A client uses a QueueConnectionFactory to create QueueConnections with a JMS point-to-point provider.

See also: ConnectionFactory and XAQueueConnectionFactory

WebSphere MQ constructor

MQQueueConnectionFactory
public MQQueueConnectionFactory()

Methods

createQueueConnection
public QueueConnection createQueueConnection()
  throws JMSException

Create a queue connection with default user identity. The connection is created in stopped mode. No messages are delivered until Connection.start method is explicitly called.

Returns:
A newly-created queue connection.

Throws:
• JMSExcetion if JMS provider fails to create queue connection because of an internal error.
• JMSSecurityException if client authentication fails because of a non valid user name or password.

createQueueConnection
public QueueConnection createQueueConnection
  (java.lang.String userName,
   java.lang.String password)
  throws JMSException

Create a queue connection with specified user identity.

Note: Use this method only with transport type JMSC.MQJMS_TP_CLIENT_MQ_TCPIP (see ConnectionFactory). The connection is created in stopped mode. No messages are delivered until Connection.start method is explicitly called.

Parameters:
• userName: the caller’s user name.
QueueConnectionFactory

- password: the caller’s password.

Returns:
A newly-created queue connection.

Throws:
- JMSException if JMS Provider fails to create queue connection because of an internal error.
- JMSSecurityException if client authentication fails because of a non valid user name or password.

getMessageRetention *
public int getMessageRetention()

Get method for messageRetention attribute.

Returns:
- JMSC.MQJMS_MRET_YES: unwanted messages remain on the input queue.
- JMSC.MQJMS_MRET_NO: unwanted messages are dealt with according to their disposition options.

getAddressReference *
public Reference getPasswordReference() throws NamingException

Return a reference for this queue connection factory.

Returns:
A reference for this object.

Throws:
NamingException.

getTemporaryModel *
public String getTemporaryModel()

getTempQPrefix *
public String getTempQPrefix()

Get the prefix that is used to form the name of a WebSphere MQ dynamic queue.

Returns:
The prefix that is used to form the name of a WebSphere MQ dynamic queue.

setMessageRetention *
pubic void setMessageRetention(int x) throws JMSException

Set method for messageRetention attribute.

Parameters:
Valid values are:
- JMSC.MQJMS_MRET_YES: unwanted messages remain on the input queue.
- JMSC.MQJMS_MRET_NO: unwanted messages are dealt with according to their disposition options. For more information on this, see “General principles for point-to-point messaging” on page 278.
setTemporaryModel *

    public void setTemporaryModel(String x) throws JMSException

setTempQPrefix *

    public void setTempQPrefix(java.lang.String tempQPrefix) throws JMSException

Set the prefix to be used to form the name of a WebSphere MQ dynamic queue.

Parameters:
    tempQPrefix: the prefix to be used to form the name of a WebSphere MQ dynamic queue.

Throws:
    JMSException if the string is null, empty, greater than 33 characters in length, or consists solely of a single asterisk (*).
QueueReceiver

public interface QueueReceiver
extends MessageConsumer

WebSphere MQ class: MQQueueReceiver

java.lang.Object
   +----com.ibm.mq.jms.MQMessageConsumer
       +----com.ibm.mq.jms.MQQueueReceiver

A client uses a QueueReceiver to receive messages that have been delivered to a queue.

See also: MessageConsumer

This class inherits the following methods from MQMessageConsumer.
• receive
• receiveNoWait
• close
• getMessageListener
• setMessageListener

Methods

getQueue

public Queue getQueue() throws JMSException

Get the queue associated with this queue receiver.

Returns:
The queue.

Throws:
JMSException if JMS fails to get queue for this queue receiver because of an internal error.
public class QueueRequestor
extends java.lang.Object

java.lang.Object
   +----javax.jms.QueueRequestor

The QueueRequestor helper class simplifies making service requests. The
QueueRequestor constructor is given a non-transacted QueueSession and a
destination Queue. It creates a TemporaryQueue for the responses, and provides a
request() method that sends the request message and waits for its reply. Users are
free to create more sophisticated versions.

See also: TopicRequestor

Constructors

QueueRequestor

   public QueueRequestor(QueueSession session,
                           Queue queue)
                          throws JMSException

This implementation assumes that the session parameter is non-transacted
and either AUTO_ACKNOWLEDGE or DUPS_OK_ACKNOWLEDGE.

Parameters:
   • session: the queue session the queue belongs to.
   • queue: the queue to perform the request/reply call on.

Throws:
   JMSException if a JMS error occurs.

Methods

close

   public void close() throws JMSException

Because a provider can allocate some resources outside the JVM on behalf
of a QueueRequestor, clients must close them when they are not needed.
You cannot rely on garbage collection to reclaim these resources eventually,
because this might not occur soon enough.

Note: This method closes the session object passed to the QueueRequestor
constructor.

Throws:
   JMSException if a JMS error occurs.

request

   public Message request(Message message)
                          throws JMSException

Send a request and wait for a reply. The temporary queue is used for
replyTo, and only one reply is expected for each request.
QueueRequestor

Parameters:

message: the message to send.

Returns:

The reply message.

Throws:

JMSException if a JMS error occurs.
public interface QueueSender extends MessageProducer

WebSphere MQ class: MQQueueSender

A client uses a QueueSender to send messages to a queue.

A QueueSender is normally associated with a particular queue. However, it is possible to create an unidentified QueueSender that is not associated with any given queue.

See also: MessageProducer

Methods

close *

public void close() throws JMSException

Because a provider can allocate some resources outside the JVM on behalf of a QueueSender, clients must close them when they are not needed. You cannot rely on garbage collection to reclaim these resources eventually, because this might not occur soon enough.

Throws: JMSException if JMS fails to close the producer because of some error.

Overrides: Close in class MQMessageProducer.

getQueue

public Queue getQueue() throws JMSException

Get the queue associated with this queue sender.

Returns: The queue.

Throws: JMSException if JMS fails to get the queue for this queue sender because of an internal error.
QueueSender

send

```java
public void send(Message message) throws JMSException
```

Send a message to the queue. Use the QueueSender’s default delivery mode, time-to-live, and priority.

**Parameters:**
- message: the message to be sent.

**Throws:**
- JMSException if JMS fails to send the message because of an error.
- MessageFormatException if a non valid message is specified.
- InvalidDestinationException if a client uses this method with a queue sender with a non valid queue.

send

```java
public void send(Message message,
                 int deliveryMode,
                 int priority,
                 long timeToLive) throws JMSException
```

Send a message specifying delivery mode, priority, and time-to-live to the queue.

**Parameters:**
- message: the message to be sent.
- deliveryMode: the delivery mode to use.
- priority: the priority for this message.
- timeToLive: the message’s lifetime (in milliseconds).

**Throws:**
- JMSException if JMS fails to send the message because of an internal error.
- MessageFormatException if a non valid message is specified.
- InvalidDestinationException if a client uses this method with a queue sender with a non valid queue.

send

```java
public void send(Queue queue,
                 Message message) throws JMSException
```

Send a message to the specified queue with the QueueSender’s default delivery mode, time-to-live, and priority.

**Note:** This method can be used only with unidentified QueueSenders.

**Parameters:**
- queue: the queue that this message should be sent to.
- message: the message to be sent.

**Throws:**
- JMSException if JMS fails to send the message because of an internal error.
- MessageFormatException if a non valid message is specified.
• InvalidDestinationException if a client uses this method with a non valid queue.

send

public void send(Queue queue,
                Message message,
                int deliveryMode,
                int priority,
                long timeToLive) throws JMSException

Send a message to the specified queue with delivery mode, priority, and time-to-live.

Note: This method can be used only with unidentified QueueSenders.

Parameters:
• queue: the queue that this message should be sent to.
• message: the message to be sent.
• deliveryMode: the delivery mode to use.
• priority: the priority for this message.
• timeToLive: the message’s lifetime (in milliseconds).

Throws:
• JMSException if JMS fails to send the message because of an internal error.
• MessageFormatException if a non valid message is specified.
• InvalidDestinationException if a client uses this method with a non valid queue.
public interface QueueSession
extends Session

WebSphere MQ class: MQQueueSession

java.lang.Object
  +----com.ibm.mq.jms.MQSession
  |    +----com.ibm.mq.jms.MQQueueSession

A QueueSession provides methods to create QueueReceivers, QueueSenders, QueueBrowsers, and TemporaryQueues.

See also: Session

The following methods are inherited from MQSession:
  • close
  • commit
  • rollback
  • recover

Methods

createBrowser
public QueueBrowser createBrowser(Queue queue)
  throws JMSException

Create a QueueBrowser to peek at the messages on the specified queue.

Parameters:
  queue: the queue to access.

Throws:
  • JMSException if a session fails to create a browser because of a JMS error.
  • InvalidDestinationException if a non valid queue is specified.

createBrowser
public QueueBrowser createBrowser(Queue queue,
                                  java.lang.String messageSelector)
  throws JMSException

Create a QueueBrowser to peek at the messages on the specified queue.

Parameters:
  • queue: the queue to access.
  • messageSelector: only deliver messages with properties that match the message selector expression.

Throws:
  • JMSException if a session fails to create a browser because of a JMS error.
  • InvalidDestinationException if a non valid queue is specified.
  • InvalidSelectorException if the message selector is not valid.
createQueue

```java
public Queue createQueue(java.lang.String queueName)
    throws JMSException
```

Create a queue with a queue name. This allows the creation of a queue with a provider-specific name. The string takes a URI format, as described on page 204.

**Note:** Clients that depend on this ability are not portable.

**Parameters:**
- `queueName`: the name of this queue.

**Returns:**
- A queue with the given name.

**Throws:**
- `JMSException` if a session fails to create a queue because of a JMS error.

createReceiver

```java
public QueueReceiver createReceiver(Queue queue)
    throws JMSException
```

Create a QueueReceiver to receive messages from the specified queue.

**Parameters:**
- `queue`: the queue to access.

**Throws:**
- `JMSException` if a session fails to create a receiver because of a JMS error.
- `InvalidDestinationException` if a non valid queue is specified.

createReceiver

```java
public QueueReceiver createReceiver(Queue queue,
                                    java.lang.String messageSelector)
    throws JMSException
```

Create a QueueReceiver to receive messages from the specified queue.

**Parameters:**
- `queue`: the queue to access.
- `messageSelector`: only messages with properties that match the message selector expression are delivered.

**Throws:**
- `JMSException` if a session fails to create a receiver because of a JMS error.
- `InvalidDestinationException` if a non valid queue is specified.
- `InvalidSelectorException` if the message selector is not valid.

createSender

```java
public QueueSender createSender(Queue queue)
    throws JMSException
```

Create a QueueSender to send messages to the specified queue.
QueueSession

Parameters:
queue: the queue to access, or null if this is to be an unidentified producer.

Throws:
• JMSException if a session fails to create a sender because of a JMS error.
• InvalidDestinationException if a non valid queue is specified.

createTemporaryQueue
public TemporaryQueue createTemporaryQueue()
throws JMSException

Create a temporary queue. Its lifetime is that of the QueueConnection unless deleted earlier.

Returns:
A temporary queue.

Throws:
JMSException if a session fails to create a temporary queue because of a JMS error.
public interface Session
extends java.lang.Runnable
Subinterfaces: QueueSession, TopicSession, XAQueueSession, XASession, and XATopicSession

WebSphere MQ class: MQSession

java.lang.object
| +----com.ibm.mq.jms.MQSession

A JMS session is a single-threaded context for producing and consuming messages.

See also: QueueSession, TopicSession, and XASession

Fields

AUTO_ACKNOWLEDGE
public static final int AUTO_ACKNOWLEDGE

With this acknowledgement mode, the session automatically acknowledges a message when it has either successfully returned from a call to receive, or the message listener it has called to process the message successfully returns.

CLIENT_ACKNOWLEDGE
public static final int CLIENT_ACKNOWLEDGE

With this acknowledgement mode, the client acknowledges a message by calling a message’s acknowledge method.

DUPS_OK_ACKNOWLEDGE
public static final int DUPS_OK_ACKNOWLEDGE

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

SESSION_TRANSACTED
public static final int SESSION_TRANSACTED

The method getAcknowledgeMode returns this value if the session is transacted and ignores the acknowledgement mode.

Methods
Session

close

public void close() throws JMSException

Because a provider can allocate some resources outside the JVM on behalf of a session, clients must close them when they are not needed. You cannot rely on garbage collection to reclaim these resources eventually, because this might not occur soon enough.

Closing a transacted session rolls back any in-progress transaction. Closing a session automatically closes its message producers and consumer, so there is no need to close them individually.

Throws:

- JMSException if the JMS implementation fails to close a session because of an internal error.

commit

public void commit() throws JMSException

Commit all messages done in this transaction and release any locks currently held.

WebSphere MQ Event Broker note

This always throws a JMSException when you have a direct connection to WebSphere MQ Event Broker.

Throws:

- JMSException if JMS fails to commit the transaction because of an internal JMS error.
- TransactionRolledBackException if the transaction gets rolled back because of an internal error during commit.
- IllegalStateException if the method is not called by a transacted session.

createBrowser (JMS 1.1 only)

public QueueBrowser createBrowser(Queue queue) throws JMSException

Create a queue browser to browse the messages on the specified queue.

Parameters:

- queue: the queue to access.

Throws:

- JMSException if the session fails to create a queue browser because of an internal JMS error.
- InvalidDestinationException if the destination is not valid.
createBrowser (JMS 1.1 only)

```java
public QueueBrowser createBrowser(Queue queue,
                                   java.lang.String messageSelector)
    throws JMSException
```

Create a queue browser to browse the messages on the specified queue using a message selector.

**Parameters:**
- queue: the queue to access.
- messageSelector: deliver only those messages with properties that match the message selector expression. A value of null or an empty string indicates that there is no message selector for the message consumer.

**Throws:**
- JMSException if the session fails to create a queue browser because of an internal JMS error.
- InvalidDestinationException if the destination is not valid.
- InvalidSelectorException if the message selector is not valid.

createBytesMessage

```java
public BytesMessage createBytesMessage() 
    throws JMSException
```

Create a BytesMessage. A BytesMessage is used to send a message containing a stream of uninterpreted bytes.

**Throws:**
- JMSException if JMS fails to create this message because of an internal error.

createConsumer (JMS 1.1 only)

```java
public MessageConsumer createConsumer(Destination destination)
    throws JMSException
```

Create a message consumer for the specified destination. Because Queue and Topic both inherit from Destination, they can be used in the destination parameter to create a message consumer.

A client uses a message consumer object to receive messages that are sent to a destination.

**Parameters:**
- destination: the destination to access.

**Throws:**
- JMSException if the session fails to create a message consumer because of an internal JMS error.
- InvalidDestinationException if the destination is not valid.
createConsumer (JMS 1.1 only)

```java
public MessageConsumer createConsumer(Destination destination,
        java.lang.String messageSelector)
        throws JMSException
```

Create a message consumer for the specified destination using a message selector. Because Queue and Topic both inherit from Destination, they can be used in the destination parameter to create a message consumer.

A client uses a message consumer object to receive messages that are sent to a destination.

**Parameters:**

- `destination`: the destination to access.
- `messageSelector`: deliver only those messages with properties that match the message selector expression. A value of null or an empty string indicates that there is no message selector for the message consumer.

**Throws:**

- `JMSException` if the session fails to create a message consumer because of an internal JMS error.
- `InvalidDestinationException` if the destination is not valid.
- `InvalidSelectorException` if the message selector is not valid.

createConsumer (JMS 1.1 only)

```java
public MessageConsumer createConsumer(Destination destination,
        java.lang.String messageSelector,
        boolean NoLocal) throws JMSException
```

Create a message consumer for the specified destination using a message selector. Because Queue and Topic both inherit from Destination, they can be used in the destination parameter to create a message consumer. If the destination is a topic, you can use this method to specify whether the consumer can receive messages published by its own connection.

A client uses a message consumer to receive messages that are published to a destination.

A connection can publish and subscribe to the same topic. The NoLocal attribute of a consumer determines whether the consumer can receive messages published by its own connection. The default value of the attribute is false.

**Parameters:**

- `destination`: the destination to access.
- `messageSelector`: deliver only those messages with properties that match the message selector expression. A value of null or an empty string indicates that there is no message selector for the message consumer.
- `NoLocal`: if true, the consumer does not receive the messages published by its own connection. The action of this parameter is defined only if the destination is a topic, not a queue.

**Throws:**

- `JMSException` if the session fails to create a message consumer because of an internal JMS error.
Session

- InvalidDestinationException if the destination is not valid.
- InvalidSelectorException if the message selector is not valid.

**createDurableSubscriber** (JMS 1.1 only)

```java
public TopicSubscriber createDurableSubscriber(Topic topic,
                                               java.lang.String name)
                                      throws JMSException
```

Create 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 uses a durable topic subscriber. The broker retains a record of this durable subscription and ensures that all messages from the publishers of the topic are retained until they are acknowledged by this durable subscriber or they expire.

Sessions with durable subscribers must always provide the same client identifier. In addition, each client must specify a name that uniquely identifies, within the client identifier, each durable subscription it creates. Only one session at a time can have a topic subscriber for a particular durable subscription.

A client can change an existing durable subscription by creating a durable topic subscriber with the same name, but with a new topic or message selector or both. Changing a durable subscriber is equivalent to unsubscribing the old one and creating a new one.

**Parameters:**
- `topic`: the topic to subscribe to. The topic must not be a temporary topic.
- `name`: the name used to identify the subscription.

**Throws:**
- JMSException if the session fails to create a subscriber because of an internal JMS error.
- InvalidDestinationException if the topic is not valid.

**createDurableSubscriber** (JMS 1.1 only)

```java
public TopicSubscriber createDurableSubscriber
                        (Topic topic,
                         java.lang.String name,
                         java.lang.String messageSelector,
                         boolean noLocal) throws JMSException
```

Create a durable subscriber to the specified topic, using a message selector and specifying whether the subscriber can receive messages published by its own connection.

If a client needs to receive all the messages published on a topic, including the ones published while the subscriber is inactive, it uses a durable topic subscriber. The broker retains a record of this durable subscription and ensures that all messages from the publishers of the topic are retained until they are acknowledged by this durable subscriber or they expire.

Sessions with durable subscribers must always provide the same client identifier. In addition, each client must specify a name which uniquely identifies, within the client identifier, each durable subscription it creates.
Only one session at a time can have a topic subscriber for a particular
durable subscription. An inactive durable subscriber is one that exists but
does not currently have a message consumer associated with it.

A client can change an existing durable subscription by creating a durable
topic subscriber with the same name, but with a new topic or message
selector or both. Changing a durable subscriber is equivalent to
unsubscribing the old one and creating a new one.

A connection can publish and subscribe to the same topic. The NoLocal
attribute of a subscriber determines whether the subscriber can receive
messages published by its own connection. The default value of the
attribute is false.

Parameters:

- topic: the topic to subscribe to. The topic must not be a
temporary topic.
- name: the name used to identify the subscription.
- messageSelector: deliver only those messages with properties
that match the message selector expression. A value of null or an
empty string indicates that there is no message selector for the
message consumer.
- noLocal: if true, the subscriber does not receive the messages
published by its own connection.

Throws:

- JMSException if the session fails to create a subscriber because of
an internal JMS error.
- InvalidDestinationException if the topic is not valid.
- InvalidSelectorException if the message selector is not valid.

createMapMessage

public MapMessage createMapMessage() throws JMSException

Create a MapMessage. A MapMessage is used to send a self-defining set of
name-value pairs, where names are strings, and values are Java primitive
types.

Throws:

- JMSException if JMS fails to create this message because of an
internal error.

createMessage

public Message createMessage() throws JMSException

Create a message. The Message interface is the root interface of all JMS
messages. It holds all the standard message header information. It can be
sent when a message containing only header information is sufficient.

Throws:

- JMSException if JMS fails to create this message because of an
internal error.
createObjectMessage

    public ObjectMessage createObjectMessage()
        throws JMSException

Create an ObjectMessage. An ObjectMessage is used to send a message that contains a serializable Java object.

Throws:
    JMSException if JMS fails to create this message because of an internal error.

createObjectMessage

    public ObjectMessage createObjectMessage
        (java.io.Serializable object)
        throws JMSException

Create an initialized ObjectMessage. An ObjectMessage is used to send a message that contains a serializable Java object.

Parameters:
    object: the object to use to initialize this message.

Throws:
    JMSException if JMS fails to create this message because of an internal error.

createProducer (JMS 1.1 only)

    public MessageProducer createProducer(Destination destination)
        throws JMSException

Create a message producer to send messages to the specified destination. Because Queue and Topic both inherit from Destination, they can be used in the destination parameter to create a message producer.

Parameters:
    destination: the destination to send messages to, or null to create a message producer that does not have a specified destination.

Throws:
    • JMSException if the session fails to create a message producer because of an internal JMS error.
    • InvalidDestinationException if the destination is not valid.

createQueue (JMS 1.1 only)

    public Queue createQueue(java.lang.String queueName) throws JMSException

Create a Queue object given a queue name. The queue name can be the name of a WebSphere MQ queue or it can be a queue URI. For information about URI format, see the “Destinations” on page 239.

This method is provided for the rare cases where a client needs to work directly with a Queue object. The client can create the Queue object with a provider specific name. Clients that depend on this ability are not portable.

The method does not create the WebSphere MQ queue. Creating a WebSphere MQ queue is an administrative task and is not done through the JMS API. The one exception is creating a temporary queue, which is done using the createTemporaryQueue() method.
Parameters:
queueName: the name of the queue.

Returns:
A Queue object with the specified name.

Throws:
JMSException if the session fails to create a queue because of an internal JMS error.

cREATE STREAM MESSAGE
public StreamMessage createStreamMessage() throws JMSException

Create a StreamMessage. A StreamMessage is used to send a self-defining stream of Java primitives.

Throws:
JMSException if JMS fails to create this message because of an internal error.

cREATE TEMPORARY QUEUE (JMS 1.1 only)
public TemporaryQueue createTemporaryQueue() throws JMSException

Create a TemporaryQueue object. The temporary queue remains until the connection ends or the queue is explicitly deleted, whichever is the sooner.

Returns:
A TemporaryQueue object.

Throws:
JMSException if the session fails to create a temporary queue because of an internal JMS error.

cREATE TEMPORARY TOPIC (JMS 1.1 only)
public TemporaryTopic createTemporaryTopic() throws JMSException

Create a TemporaryTopic object. The temporary topic remains until the connection ends or the topic is explicitly deleted, whichever is the sooner.

Returns:
A TemporaryTopic object.

Throws:
JMSException if the session fails to create a temporary topic because of an internal JMS error.

CREATE TEXT MESSAGE
public TextMessage createTextMessage() throws JMSException

Create a TextMessage. A TextMessage is used to send a message containing a string.

Throws:
JMSException if JMS fails to create this message because of an internal error.
createTextMessage

```java
public TextMessage createTextMessage(java.lang.String string)
throws JMSException

Create an initialized TextMessage. A TextMessage is used to send a message containing a string.

Parameters:
  string: the string used to initialize this message.

Throws:
  JMSException if JMS fails to create this message because of an internal error.
```

createTopic (JMS 1.1 only)

```java
public Topic createTopic(java.lang.String topicName) throws JMSException

Create a Topic object given a topic name. The topic name can be the name of a broker topic or it can be a topic URI. For information about URI format, see “Destinations” on page 239.

This method is provided for the rare cases where a client needs to work directly with a Topic object. The client can create the Topic object with a provider specific name. Clients that depend on this ability are not portable.

This method does not create the broker topic. Creating a broker topic is an administrative task and is not done through the JMS API. The one exception is creating a temporary topic, which is done using the createTemporaryTopic() method.

Parameters:
  topicName: the name of the topic.

Returns:
  A Topic object with the specified name.

Throws:
  JMSException if the session fails to create a topic because of an internal JMS error.
```

getAcknowledgeMode (JMS 1.1 only)

```java
public int getAcknowledgeMode() throws JMSException

Return the acknowledgement mode for the session. The acknowledgement mode is specified when the session is created. A session that is transacted has no acknowledgement mode.

Returns:
  The acknowledgement mode for the session, provided the session is not transacted. If the session is transacted, the method returns SESSION_TRANSACTED.

Throws:
  JMSException if JMS fails to return the acknowledgment mode because of an internal JMS error.
```

See also:
  Connection.createSession
Session

**getMessageListener**

```java
public MessageListener getMessageListener() throws JMSException
```

Return the session’s distinguished message listener.

**Returns:**
The message listener associated with this session.

**Throws:**
JMSException if JMS fails to get the message listener because of an internal JMS error.

**See also:**
setMessageListener

**getTransacted**

```java
public boolean getTransacted() throws JMSException
```

Whether the session is in transacted mode.

**WebSphere MQ Event Broker note**
This method always returns false when you have a direct connection to WebSphere MQ Event Broker.

**Returns:**
True if the session is in transacted mode.

**Throws:**
JMSException if JMS fails to return the transaction mode because of an internal error in JMS Provider.

**recover**

```java
public void recover() throws JMSException
```

Stop the delivery of messages in this session, and restart the sending messages with the oldest unacknowledged message.

**WebSphere MQ Event Broker note**
This always throws a JMSException when you have a direct connection to WebSphere MQ Event Broker.

**Throws:**
- JMSException if JMS fails to stop the delivery of messages and restart the sending messages because of an internal JMS error.
- IllegalStateException if the method is called by a transacted session.
rollback

public void rollback() throws JMSException

Roll back any messages done in this transaction and release any locks currently held.

---

**WebSphere MQ Event Broker note**

This always throws a JMSException when you have a direct connection to WebSphere MQ Event Broker.

Throws:

- JMSException if JMS fails to roll back the transaction because of an internal JMS error.
- IllegalStateException if the method is not called by a transacted session.

---

run

public void run()

This method is intended for use only by application servers.

---

**WebSphere MQ Event Broker note**

This always throws an IllegalStateException when you have a direct connection to WebSphere MQ Event Broker.

Specified by:

- run in the interface java.lang.Runnable

See also:

- ServerSession

setMessageListener

public void setMessageListener(MessageListener listener) throws JMSException

Set the session’s distinguished message listener. When it is set, no other form of message receipt in the session can be used. However, all forms of sending messages are still supported.

This is an expert facility that is not used by regular JMS clients.

Parameters:

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

Throws:

- JMSException if JMS fails to set the message listener because of an internal error in the JMS Provider.

See also:

- getMessageListener
unsubscribe (JMS 1.1 only)

    public void unsubscribe(java.lang.String name) throws JMSException

    Unsubscribe a durable subscription that has been created by a client.

    This method tells the broker that the durable subscription has ended and
    not to send any more messages to the subscriber.

    It is not advisable for a client to delete a durable subscription while there
    is an active message consumer 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 the subscription.

    Throws:
    • JMSException if the session fails to remove the durable
      subscription because of an internal JMS error.
    • InvalidDestinationException if the subscription name is not
      valid.

    Note
    For a direct connection to WebSphere MQ Event Broker, WebSphere
    Business Integration Event Broker, or WebSphere Business Integration
    Message Broker, this method throws a JMSException.
StreamMessage

public interface StreamMessage extends Message

WebSphere MQ class: JMSStreamMessage

java.lang.Object
   +----com.ibm.jms.JMSMessage
   |    +----com.ibm.jms.JMSStreamMessage

Use a StreamMessage to send a stream of Java primitives.

See also: BytesMessage, MapMessage, Message, ObjectMessage and TextMessage

Methods

readBoolean

public boolean readBoolean() throws JMSException

Read a boolean from the stream message.

Returns:
   The boolean value read.

Throws:
   • JMSException if JMS fails to read the message because of an internal JMS error.
   • MessageEOFException if an end of message stream is received.
   • MessageFormatException if this type conversion is not valid.
   • MessageNotReadableException if the message is in write-only mode.

readByte

public byte readByte() throws JMSException

Read a byte value from the stream message.

Returns:
   The next byte from the stream message as an 8-bit byte.

Throws:
   • JMSException if JMS fails to read the message because of an internal JMS error.
   • MessageEOFException if an end of message stream is received.
   • MessageFormatException if this type conversion is not valid.
   • MessageNotReadableException if the message is in write-only mode.
readBytes

```java
public int readBytes(byte[] value)
    throws JMSException
```

Read a byte array field from the stream message into the specified byte[] object (the read buffer). If the buffer size is less than, or equal to, the size of the data in the message field, an application must make further calls to this method to retrieve the remainder of the data. Once the first readBytes call on a byte[] field value has been done, 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 throws a MessageFormatException.

**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 byte field has been reached.

**Throws:**

- JMSException if JMS fails to read the message because of an internal JMS error.
- MessageEOFException if an end of message stream is received.
- MessageFormatException if this type conversion is not valid.
- MessageNotReadableException if the message is in write-only mode.

readChar

```java
public char readChar() throws JMSException
```

Read a Unicode character value from the stream message.

**Returns:**

A Unicode character from the stream message.

**Throws:**

- JMSException if JMS fails to read the message because of an internal JMS error.
- MessageEOFException if an end of message stream is received.
- MessageFormatException if this type conversion is not valid.
- MessageNotReadableException if the message is in write-only mode.

readDouble

```java
public double readDouble() throws JMSException
```

Read a double from the stream message.

**Returns:**

A double value from the stream message.

**Throws:**

- JMSException if JMS fails to read the message because of an internal JMS error.
- MessageEOFException if an end of message stream is received.
- MessageFormatException if this type conversion is not valid.
• MessageNotReadableException if the message is in write-only mode.

**readFloat**

```java
class StreamMessage {
    public float readFloat() throws JMSException {
        ...  // Read a float from the stream message.
    }
}
```

**Returns:**

A float value from the stream message.

**Throws:**

• JMSException if JMS fails to read the message because of an internal JMS error.
• MessageEOFException if an end of message stream
• MessageFormatException if this type conversion is not valid.
• MessageNotReadableException if the message is in write-only mode.

**readInt**

```java
class StreamMessage {
    public int readInt() throws JMSException {
        ...  // Read a 32-bit integer from the stream message.
    }
}
```

**Returns:**

A 32-bit integer value from the stream message, interpreted as an int.

**Throws:**

• JMSException if JMS fails to read the message because of an internal JMS error.
• MessageEOFException if an end of message stream is received.
• MessageFormatException if this type conversion is not valid.
• MessageNotReadableException if the message is in write-only mode.

**readLong**

```java
class StreamMessage {
    public long readLong() throws JMSException {
        ...  // Read a 64-bit integer from the stream message.
    }
}
```

**Returns:**

A 64-bit integer value from the stream message, interpreted as a long.

**Throws:**

• JMSException if JMS fails to read the message because of an internal JMS error.
• MessageEOFException if an end of message stream
• MessageFormatException if this type conversion is not valid.
• MessageNotReadableException if the message is in write-only mode.
StreamMessage

readObject

public java.lang.Object readObject() throws JMSException

Read a Java object from the stream message.

Returns:
A Java object from the stream message in object format (for example, if it was set as an int, an integer is returned).

Throws:
• JMSException if JMS fails to read the message because of an internal JMS error.
• MessageEOFException if an end of message stream is received.
• NotReadableException if the message is in write-only mode.

readShort

public short readShort() throws JMSException

Read a 16-bit number from the stream message.

Returns:
A 16-bit number from the stream message.

Throws:
• JMSException if JMS fails to read the message because of an internal JMS error.
• MessageEOFException if an end of message stream is received.
• MessageFormatException if this type conversion is not valid.
• MessageNotReadableException if the message is in write-only mode.

readString

public java.lang.String readString() throws JMSException

Read in a string from the stream message.

Returns:
A Unicode string from the stream message.

Throws:
• JMSException if JMS fails to read the message because of an internal JMS error.
• MessageEOFException if an end of message stream is received.
• MessageFormatException if this type conversion is not valid.
• MessageNotReadableException if the message is in write-only mode
StreamMessage

reset

public void reset() throws JMSException

Put the message in read-only mode, and reposition the stream to the beginning.

Throws:
  • JMSException if JMS fails to reset the message because of an internal JMS error.
  • MessageFormatException if the message has an non valid format.

writeBoolean

public void writeBoolean(boolean value) throws JMSException

Write a boolean to the stream message.

Parameters:
  value: the boolean value to be written.

Throws:
  • JMSException if JMS fails to read the message because of an internal JMS error.
  • MessageNotWriteableException if the message is in read-only mode.

writeByte

public void writeByte(byte value) throws JMSException

Write a byte to the stream message.

Parameters:
  value: the byte value to be written.

Throws:
  • JMSException if JMS fails to write the message because of an internal JMS error.
  • MessageNotWriteableException if the message is in read-only mode.

writeBytes

public void writeBytes(byte[] value) throws JMSException

Write a byte array to the stream message.

Parameters:
  value: the byte array to be written.

Throws:
  • JMSException if JMS fails to write the message because of an internal JMS error.
  • MessageNotWriteableException if the message is in read-only mode.
StreamMessage

writeBytes

public void writeBytes(byte[] value,
                       int offset,
                       int length) throws JMSException

Write a portion of a byte array to the stream message.

Parameters:

- value: the byte array value to be written.
- offset: the initial offset within the byte array.
- length: the number of bytes to use.

Throws:

- JMSException if JMS fails to write the message because of an internal JMS error.
- MessageNotWriteableException if the message is in read-only mode.

writeChar

public void writeChar(char value) throws JMSException

Write a character to the stream message.

Parameters:

- value: the character value to be written.

Throws:

- JMSException if JMS fails to write the message because of an internal JMS error.
- MessageNotWriteableException if the message is in read-only mode.

writeDouble

public void writeDouble(double value) throws JMSException

Write a double to the stream message.

Parameters:

- value: the double value to be written.

Throws:

- JMSException if JMS fails to write the message because of an internal JMS error.
- MessageNotWriteableException if the message is in read-only mode.

writeFloat

public void writeFloat(float value) throws JMSException

Write a float to the stream message.

Parameters:

- value: the float value to be written.

Throws:

- JMSException if JMS fails to write the message because of an internal JMS error.
StreamMessage

- MessageNotWriteableException if the message is in read-only mode.

**writeInt**

```java
public void writeInt(int value) throws JMSException
```

Write an integer to the stream message.

**Parameters:**
value: the integer to be written.

**Throws:**
- JMSException if JMS fails to write the message because of an internal JMS error.
- MessageNotWriteableException if the message is in read-only mode.

**writeLong**

```java
public void writeLong(long value) throws JMSException
```

Write a long to the stream message.

**Parameters:**
value: the long to be written.

**Throws:**
- JMSException if JMS fails to write the message because of an internal JMS error.
- MessageNotWriteableException if the message is in read-only mode.

**writeObject**

```java
public void writeObject(java.lang.Object value) throws JMSException
```

Write a Java object to the stream message. This method works only for object primitive types (for example, Integer, Double, Long), strings, and byte arrays.

**Parameters:**
value: the Java object to be written.

**Throws:**
- JMSException if JMS fails to write the message because of an internal JMS error.
- MessageNotWriteableException if the message is in read-only mode.
- MessageFormatException if the object is not valid.
StreamMessage

writeShort

public void writeShort(short value) throws JMSException

Write a short to the stream message.

Parameters:
value: the short to be written.

Throws:
• JMSException if JMS fails to write the message because of an internal JMS error.
• MessageNotWriteableException if the message is in read-only mode.

writeString

public void writeString(java.lang.String value) throws JMSException

Write a string to the stream message.

Parameters:
value: the string value to be written.

Throws:
• JMSException if JMS fails to write the message because of an internal JMS error.
• MessageNotWriteableException if the message is in read-only mode.
TemporaryQueue

public interface TemporaryQueue
extends Queue

WebSphere MQ class: MQTemporaryQueue

java.lang.Object
  +----com.ibm.mq.jms.MQDestination
  |      +----com.ibm.mq.jms.MQQueue
  |      |      +----com.ibm.mq.jms.MQTemporaryQueue

A TemporaryQueue is a unique queue object that is created for the duration of a QueueConnection.

Methods

delete

    public void delete() throws JMSException

    Delete this temporary queue. If there are still existing senders or receivers using it, a JMSException is thrown.

    Throws:
    JMSException if JMS implementation fails to delete a TemporaryQueue because of an internal error.
TemporaryTopic

public interface TemporaryTopic extends Topic

WebSphere MQ class: MQTemporaryTopic

java.lang.Object
   +----com.ibm.mq.jms.MQDestination
       +----com.ibm.mq.jms.MQTopic
           +----com.ibm.mq.jms.MQTemporaryTopic

A TemporaryTopic is a unique topic object created for the duration of a TopicConnection and can be consumed only by consumers of that connection.

WebSphere MQ constructor

MQTemporaryTopic
MQTemporaryTopic() throws JMSException

Methods

delete

public void delete() throws JMSException

Delete this temporary topic. If there are still existing publishers or subscribers using it, a JMSException is thrown.

Throws:
   JMSException if JMS implementation fails to delete a TemporaryTopic because of an internal error.
public interface TextMessage
extends Message

WebSphere MQ class: JMSTextMessage

java.lang.Object
  +---com.ibm.jms.JMSMessage
  |    +---com.ibm.jms.JMSTextMessage

Use TextMessage to send a message containing a java.lang.String. It inherits from Message and adds a text message body.

See also: BytesMessage, MapMessage, Message, ObjectMessage and StreamMessage

Methods

getText

public java.lang.String getText() throws JMSException

Get the string containing this message’s data. The default value is null.

Returns:
  The string containing the message’s data.

Throws:
  JMSException if JMS fails to get the text because of an internal JMS error.

setText

public void setText(java.lang.String string)
  throws JMSException

Set the string containing this message’s data.

Parameters:
  string: the string containing the message’s data.

Throws:
  • JMSEception if JMS fails to set text because of an internal JMS error.
  • MessageNotWriteableException if the message is in read-only mode.
public interface Topic
extends Destination
Subinterfaces: TemporaryTopic

WebSphere MQ class: MQTopic

java.lang.Object
   \---com.ibm.mq.jms.MQDestination
      \---com.ibm.mq.jms.MQTopic

A Topic object encapsulates a provider-specific topic name. It is the way that a client specifies the identity of a topic to JMS methods.

### WebSphere MQ Event Broker note
For direct connections to WebSphere MQ Event Broker, properties accessed by methods marked with a § are ignored.

See also: Destination

### WebSphere MQ constructor

MQTopic
   public MQTopic() 
   public MQTopic(string URItopic)

See TopicSession.createTopic.

### Methods

**getBaseTopicName** *

   public String getBaseTopicName()

Get method for the underlying WebSphere MQ topic name.

**getBrokerCCDurSubQueue** §

   public String getBrokerCCDurSubQueue()

Get method for brokerCCDurSubQueue attribute.

**Returns:**

The name of the durable subscription queue (the brokerCCDurSubQueue) to use for a ConnectionConsumer.
getBrokerDurSubQueue * §

public String getBrokerDurSubQueue()

Get method for brokerDurSubQueue attribute.

Returns:
The name of the durable subscription queue (the brokerDurSubQueue) to use.

getBrokerVersion *

public int getBrokerVersion()

Get method for brokerVersion attribute.

Returns:
The broker's version number

getMulticast *

public int getMulticast()

Get method for the multicast attribute.

Returns:
An integer representing the current multicast setting.

See also:
setMulticast()

ggetReference *

public Reference getReference()

Create a reference for this topic.

Returns:
A reference for this object.

Throws:
NamingException.

getTopicName

public java.lang.String getTopicName() throws JMSException

Get the name of this topic in URI format. (URI format is described in "Creating topics at runtime" on page 223.) For information specific to JMS 1.1, see "Destinations" on page 239.

Note: Clients that depend upon the name are not portable.

Returns:
The topic name.

Throws:
JMSException if JMS implementation for topic fails to return the topic name because of an internal error.
setBaseTopicName *
public void setBaseTopicName(String x)

Set method for the underlying WebSphere MQ topic name.

setBrokerCCDurSubQueue * §
public void setBrokerCCDurSubQueue(String x) throws JMSException

Set method for brokerCCDurSubQueue attribute.

Parameters:
brokerCCDurSubQueue: the name of the durable subscription queue to use for a ConnectionConsumer.

setBrokerDurSubQueue * §
public void setBrokerDurSubQueue(String x) throws JMSException

Set method for brokerDurSubQueue attribute.

Parameters:
brokerDurSubQueue: the name of the durable subscription queue to use.

setBrokerVersion *
public void setBrokerVersion(int x) throws JMSException

Set method for brokerVersion attribute.

Parameters:
An integer representing one of the valid broker version number values. These are represented by the constants:
JMSC.MQJMS_BROKER_V1
JMSC.MQJMS_BROKER_V2

setMulticast *
public void setMulticast(int x) throws JMSException

Set method for the multicast attribute.

Parameters:
x: an integer specifying a multicast setting. The following are symbolic constants that represent the valid values of the parameter:
JMSC.MQJMS_MULTICAST_AS_CF
JMSC.MQJMS_MULTICAST_DISABLED
JMSC.MQJMS_MULTICAST_NOT_RELIABLE
JMSC.MQJMS_MULTICAST_RELIABLE
JMSC.MQJMS_MULTICAST_ENABLED

Throws:
JMSException if the parameter does not represent a valid multicast setting.
toString

public String toString()

Return a well-formatted printed version of the topic name.

Returns:
The provider-specific identity values for this topic.

Overrides:
toString in class Object.
public interface TopicConnection
extends Connection
Subinterfaces: XATopicConnection

WebSphere MQ class: MQTopicConnection

A TopicConnection is an active connection to a JMS publish/subscribe provider.

See also: Connection, TopicConnectionFactory, and XATopicConnection

Methods

createConnectionConsumer

public ConnectionConsumer createConnectionConsumer
     (Topic topic,
      java.lang.String messageSelector,
      ServerSessionPool sessionPool,
      int maxMessages)
     throws JMSException

Create a connection consumer for this connection. This is an expert facility that is not used by regular JMS clients.

--- WebSphere MQ Event Broker note ---
For a direct connection to WebSphere MQ Event Broker, this method throws a JMSException.

Parameters:

- topic: the topic to access.
- messageSelector: only deliver messages with properties that match the message selector expression.
- 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.

Throws:

- JMSException if the JMS Connection fails to create a connection consumer because of an internal error, or because of incorrect arguments for sessionPool.
- InvalidDestinationException if the topic is not valid.
- InvalidSelectorException if the message selector is not valid.

See also:
ConnectionConsumer
createDurableConnectionConsumer

public ConnectionConsumer createDurableConnectionConsumer
    (Topic topic,
     java.lang.String subscriptionName
     java.lang.String messageSelector,
     ServerSessionPool sessionPool,
     int maxMessages)
    throws JMSException

Create a durable connection consumer for this connection. This is an expert facility that is not used by regular JMS clients.

Parameters:
- topic: the topic to access.
- subscriptionName: the name of the durable subscription.
- messageSelector: deliver only messages with properties that match the message selector expression.
- sessionPool: the server session pool to associate with this durable connection consumer.
- maxMessages: the maximum number of messages that can be assigned to a server session at one time.

Returns:
The durable connection consumer.

Throws:
- JMSException if the JMS Connection fails to create a connection consumer because of an internal error, or because of incorrect arguments for sessionPool and messageSelector.
- InvalidDestinationException if the topic is not valid.
- InvalidSelectorException if the message selector is not valid.

See also:
ConnectionConsumer

createTopicSession

public TopicSession createTopicSession(boolean transacted,
                                       int acknowledgeMode)
                                      throws JMSException

Create a TopicSession.

Parameters:
- transacted: if true, the session is transacted.
- acknowledgeMode: one of:
  Session.AUTO_ACKNOWLEDGE
TopicConnection

Session.CLIENT_ACKNOWLEDGE
Session.DUPS_OK_ACKNOWLEDGE

Indicates whether the consumer or the client acknowledge any messages that they receive. This parameter is ignored if the session is transacted.

**Returns:**

A newly-created topic session.

**Throws:**

JMSException if JMS Connection fails to create a session because of an internal error, or a lack of support for the specific transaction and acknowledgement mode.
TopicConnectionFactory

public interface TopicConnectionFactory
extends ConnectionFactory
Subinterfaces: XATopicConnectionFactory

WebSphere MQ class: MQTopicConnectionFactory

java.lang.Object
   +----com.ibm.mq.jms.MQConnectionFactory
      +----com.ibm.mq.jms.MQTopicConnectionFactory

A client uses a TopicConnectionFactory to create TopicConnections with a JMS publish/subscriber provider.

Note
For direct connections to WebSphere MQ Event Broker, WebSphere Business Integration Event Broker, or WebSphere Business Integration Message Broker, properties accessed by methods marked with a § are ignored.

See also: ConnectionFactory and XATopicConnectionFactory

WebSphere MQ constructor

MQTopicConnectionFactory

public MQTopicConnectionFactory()

Methods

createTopicConnection

public TopicConnection createTopicConnection()
throws JMSException

Create a topic connection with default user identity. The connection is created in stopped mode. No messages are delivered until Connection.start method is explicitly called.

Returns:
A newly-created topic connection.

Throws:

- JMSException if JMS Provider fails to create a Topic Connection because of an internal error.
- JMSSecurityException if client authentication fails because of a non valid user name or password.
createTopicConnection

public TopicConnection createTopicConnection(
  java.lang.String userName,
  java.lang.String password)
throws JMSException

Create a topic connection with specified user identity. The connection is
created in stopped mode. No messages are delivered until Connection.start
method is explicitly called.

Parameters:
  • userName: the caller's user name.
  • password: the caller's password.

Returns:
  A newly-created topic connection.

Throws:
  • JMSEException if JMS Provider fails to create a Topic Connection
    because of an internal error.
  • JMSSecurityException if client authentication fails because of a
    non valid user name or password.

Note: This method is valid only for transport type
IBM_JMS_TP_CLIENT_MQ_TCPIP. See ConnectionFactory.

getBrokerCCSubQueue *

public String getBrokerCCSubQueue()

Get method for brokerCCSubQueue attribute.

Returns:
  The name of the nondurable subscription queue to use for a
  connection consumer.

getBrokerControlQueue *

public String getBrokerControlQueue()

Get method for brokerControlQueue attribute.

Returns:
  The broker's control queue name.

getBrokerPubQueue *

public String getBrokerPubQueue()

Get method for brokerPubQueue attribute.

Returns:
  The broker's publish queue name.
public String getBrokerQueueManager()

Get method for brokerQueueManager attribute.

Returns:
The broker's queue manager name.

public String getBrokerSubQueue()

Get method for brokerSubQueue attribute.

Returns:
The name of the nondurable subscription queue to use.

public int getBrokerVersion()

Get method for brokerVersion attribute.

Returns:
The broker's version number.

public long getCleanupInterval()

Get method for cleanupInterval attribute.

Returns:
How often the cleanup utility runs, in milliseconds

public int getCleanupLevel()

Get method for cleanupLevel attribute.

Returns:
The value of cleanupLevel

public int getDirectAuth()

Get method for the direct authentication attribute.

Returns:
The value of the direct authentication attribute

See also:
setDirectAuth()
TopicConnectionFactory

- `getMessageSelection * §`  
  `public int getMessageSelection()`  
  Get method for the message selection attribute.  
  **Returns:**  
  The value of the message selection attribute  
  **See also:**  
  `setMessageSelection()`  

- `getMulticast *`  
  `public int getMulticast()`  
  Get method for the multicast attribute.  
  **Returns:**  
  An integer representing the current multicast setting.  
  **See also:**  
  `setMulticast()`  

- `getProxyHostName *`  
  `public String getProxyHostName()`  
  Get method for the proxy host name attribute.  
  **Returns:**  
  The host name of the proxy server when establishing a direct connection, or null if no proxy server is used.  

- `getProxyPort *`  
  `public int getProxyPort()`  
  Get method for the proxy port attribute.  
  **Returns:**  
  The port number to connect to on the proxy server.  

- `getPubAckInterval * §`  
  `public int getPubAckInterval()`  
  Get method for pubAckInterval attribute.  
  **Returns:**  
  The interval, in number of messages, between publish requests that require acknowledgement from the broker.  

- `getReference *`  
  `public Reference getReference()`  
  Return a reference for this topic connection factory.  
  **Returns:**  
  A reference for this topic connection factory.  
  **Throws:**  
  NamingException.
getSparseSubscriptions *
public boolean getSparseSubscriptions()

Get method for the sparse subscriptions attribute.

Returns:
The value of the sparse subscriptions attribute

See also:
setSparseSubscriptions()

getStatusRefreshInterval * §
public int getStatusRefreshInterval()

Get method for statusRefreshInterval attribute.

Returns:
The number of milliseconds between transactions to refresh publish/subscribe status.

getSubscriptionStore * §
public int getSubscriptionStore()

Get method for the SUBSTORE property.

Returns:
An integer representing the current SUBSTORE property.

setBrokerCCSubQueue * §
public void setBrokerCCSubQueue(String x) throws JMSException

Set method for brokerCCSubQueue attribute.

Parameters:
brokerSubQueue: the name of the nondurable subscription queue to use for a connection consumer.

setBrokerControlQueue * §
public void setBrokerControlQueue(String x) throws JMSException

Set method for brokerControlQueue attribute.

Parameters:
brokerControlQueue: the name of the broker control queue.

setBrokerPubQueue * §
public void setBrokerPubQueue(String x) throws JMSException

Set method for brokerPubQueue attribute.

Parameters:
brokerPubQueue: the name of the broker publish queue.
setBrokerQueueManager * §
   public void setBrokerQueueManager(String x) throws JMSException

   Set method for brokerQueueManager attribute.

   Parameters:
   brokerQueueManager: the name of the broker’s queue manager.

setBrokerSubQueue * §
   public void setBrokerSubQueue(String x) throws JMSException

   Set method for brokerSubQueue attribute.

   Parameters:
   brokerSubQueue: the name of the nondurable subscription queue to use.

setBrokerVersion *
   public void setBrokerVersion(int x) throws JMSException

   Set method for brokerVersion attribute.

   Parameters:
   An integer representing one of the valid broker version number values. These are represented by the constants:
   JMSC.MQJMS_BROKER_V1
   JMSC.MQJMS_BROKER_V2

setCleanupInterval * §
   public void setCleanupInterval(long x) throws JMSException

   Set method for cleanupInterval attribute.

   Parameters:
   How often the cleanup utility runs, in milliseconds

setCleanupLevel * §
   public void setCleanupLevel(int x) throws JMSException

   Set method for cleanupLevel attribute.

   Parameters:
   An integer representing one of the valid cleanup levels. These are represented by the constants:
   JMSC.MQJMS_CLEANUP_NONE
   JMSC.MQJMS_CLEANUP_SAFE
   JMSC.MQJMS_CLEANUP_STRONG
   JMSC.MQJMS_CLEANUP_AS_PROPERTY
setDirectAuth *

```java
public void setDirectAuth(int x) throws JMSException
```

Set method for the direct authentication attribute.

**Parameters:**

- `x`: an integer specifying the type of direct authentication that is required. The following are symbolic constants that represent the valid values of the parameter:
  - `JMSC.MQJMS_DIRECTAUTH_BASIC`
  - `JMSC.MQJMS_DIRECTAUTH_CERTIFICATE`

setMessageSelection * §

```java
public void setMessageSelection(int x)
```

Set method for the message selection attribute.

**Parameters:**

- `x`: an integer indicating whether the client or the broker performs message selection. The following are symbolic constants that represent the valid values of the parameter:
  - `JMSC.MQJMS_MSEL_CLIENT`
  - `JMSC.MQJMS_MSEL_BROKER`

setMulticast *

```java
public void setMulticast(int x) throws JMSException
```

Set method for the multicast attribute.

**Parameters:**

- `x`: an integer specifying a multicast setting. The following are symbolic constants that represent the valid values of the parameter:
  - `JMSC.MQJMS_MULTICAST_DISABLED`
  - `JMSC.MQJMS_MULTICAST_NOT_RELIABLE`
  - `JMSC.MQJMS_MULTICAST_RELIABLE`
  - `JMSC.MQJMS_MULTICAST_ENABLED`

setProxyHostName *

```java
public void setProxyHostName(String proxyHostName) throws JMSException
```

Set method for the proxy host name attribute.

**Parameters:**

- `proxyHostName`: the host name of the proxy server when establishing a direct connection, or null if no proxy server is used.

setProxyPort *

```java
public void setProxyPort(int proxyPort) throws JMSException
```

Set method for the proxy port attribute.

**Parameters:**

- `proxyPort`: the port number to connect to on the proxy server.
setPubAckInterval * §

public void setPubAckInterval(int x)

Set method for pubAckInterval attribute. The number of messages to publish between requiring acknowledgement from the broker. The default is 25. Applications do not normally alter this value, and must not rely on this acknowledgement.

Parameters:

  pubAckInterval: the number of messages to use as an interval.

setSparseSubscriptions *

public void setSparseSubscriptions(boolean x)

Set method for the sparse subscriptions attribute. A sparse subscription is one that receives infrequent matching messages. The default value of this attribute is false. A value of true might be required if an application using sparse subscriptions fails to receive messages because of log overflow. If you set the attribute to true, the application must be able to open the subscriber queue for browsing messages.

Parameters:

  x: indicates whether sparse subscriptions are selected.

setStatusRefreshInterval * §

public void setStatusRefreshInterval(int x)

Set method for statusRefreshInterval attribute.

Parameters:

  statusRefreshInterval: the number of milliseconds between transactions to refresh publish/subscribe status.

setSubscriptionStore * §

public void setSubscriptionStore(int x) throws JMSException

Set method for the SUBSTORE property.

Parameters:

  SubStoretype: an integer representing one of the valid values of the SUBSTORE property. The following symbolic constants represent the valid values:

  JMSC.MQJMS_SUBSTORE_QUEUE
  JMSC.MQJMS_SUBSTORE_BROKER
  JMSC.MQJMS_SUBSTORE_MIGRATE
TopicPublisher

```java
public interface TopicPublisher
extends MessageProducer
```

WebSphere MQ class: MQTopicPublisher

```java
java.lang.Object  
+----com.ibm.mq.jms.MQMessageProducer  
     +----com.ibm.mq.jms.MQTopicPublisher
```

A client uses a TopicPublisher for publishing messages on a topic. TopicPublisher is the publish/subscribe variant of a JMS message producer.

## Methods

close *

```java
public void close() throws JMSException
```

Because a provider can allocate some resources outside the JVM on behalf of a TopicPublisher, clients must close them when they are not needed. You cannot rely on garbage collection to reclaim these resources eventually, because this might not occur soon enough.

**Throws:**

JMSException if JMS fails to close the producer because of an error.

**Overrides:**

close in class MQMessageProducer.

getTopic

```java
public Topic getTopic() throws JMSException
```

Get the topic associated with this publisher.

**Returns:**

This publisher’s topic

**Throws:**

JMSException if JMS fails to get the topic for this topic publisher because of an internal error.

publish

```java
public void publish(Message message) throws JMSException
```

Publish a message to the topic. Use the topic’s default delivery mode, time-to-live, and priority.

**Parameters:**

message: the message to publish

**Throws:**

- JMSException if JMS fails to publish the message because of an internal error.
- MessageFormatException if a non valid message is specified.
- InvalidDestinationException if a client uses this method with a Topic Publisher with a non valid topic.
TopicPublisher

```java
public void publish(Message message,
                     int deliveryMode,
                     int priority,
                     long timeToLive) throws JMSException
```

Publish a message to the topic specifying delivery mode, priority, and time-to-live to the topic.

**WebSphere MQ Event Broker note**

If `deliveryMode` is `PERSISTENT` or `timeToLive` is greater than 0, this method throws a `JMSException` when you have a direct connection to WebSphere MQ Event Broker.

**Parameters:**
- `message`: the message to publish.
- `deliveryMode`: the delivery mode to use.
- `priority`: the priority for this message.
- `timeToLive`: the message’s lifetime (in milliseconds).

**Throws:**
- `JMSException` if JMS fails to publish the message because of an internal error.
- `MessageFormatException` if a non valid message is specified.
- `InvalidDestinationException` if a client uses this method with a Topic Publisher with a non valid topic.

```java
public void publish(Topic topic,
                    Message message) throws JMSException
```

Publish a message to a topic for an unidentified message producer. Use the topic’s default delivery mode, time-to-live, and priority.

**Parameters:**
- `topic`: the topic to publish this message to.
- `message`: the message to send.

**Throws:**
- `JMSException` if JMS fails to publish the message because of an internal error.
- `MessageFormatException` if a non valid message is specified.
- `InvalidDestinationException` if a client uses this method with a non valid topic.
public void publish(Topic topic,
    Message message,
    int deliveryMode,
    int priority,
    long timeToLive) throws JMSException

Publish a message to a topic for an unidentified message producer, specifying delivery mode, priority, and time-to-live.

**WebSphere MQ Event Broker note**

If deliveryMode is PERSISTENT or timeToLive is greater than 0, this method throws a JMSException when you have a direct connection to WebSphere MQ Event Broker.

**Parameters:**

- **topic**: the topic to publish 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).

**Throws:**

- **JMSException** if JMS fails to publish the message because of an internal error.
- **MessageFormatException** if a non valid message is specified.
- **InvalidDestinationException** if a client uses this method with a non valid topic.
public class TopicRequestor extends java.lang.Object

JMS provides this TopicRequestor class to assist with making service requests.

The TopicRequestor constructor is given a non-transacted TopicSession and a destination Topic. It creates a TemporaryTopic for the responses, and provides a request() method that sends the request message and waits for its reply. Users are free to create more sophisticated versions.

Constructors

Constructor for the TopicRequestor class. This implementation assumes that the session parameter is non-transacted, and either AUTO_ACKNOWLEDGE or DUPS_OK_ACKNOWLEDGE.

Parameters:
- session: the topic session the topic belongs to.
- topic: the topic to perform the request/reply call on.

Throws:
   JMSException if a JMS error occurs.

Methods

Because a provider can allocate some resources outside the JVM on behalf of a TopicRequestor, clients must close them when they are not needed. You cannot rely on garbage collection to reclaim these resources eventually, because this might not occur soon enough.

Note: This method closes the session object passed to the TopicRequestor constructor.

Throws:
   JMSException if a JMS error occurs.

Send a request and wait for a reply.

Parameters:
- message: the message to send.
Returns:
The reply message.

Throws:
JMSException if a JMS error occurs.
TopicSession

public interface TopicSession
extends Session

WebSphere MQ class: MQTopicSession

java.lang.Object
  +----com.ibm.mq.jms.MQSession
      +----com.ibm.mq.jms.MQTopicSession


See also: Session

WebSphere MQ constructor

MQTopicSession

public MQTopicSession(boolean transacted,
                        int acknowledgeMode) throws JMSException

See TopicConnection.createTopicSession.

Methods

createDurableSubscriber

public TopicSubscriber createDurableSubscriber
(Topic topic,
    java.lang.String name) throws JMSException

Create a durable subscriber to the specified topic.

WebSphere MQ Event Broker note

This method throws a JMSException when you have a direct connection to WebSphere MQ Event Broker.

Parameters:

- topic: the topic to subscribe to.
- name: the name used to identify this subscription.

Throws:

- JMSException if a session fails to create a subscriber because of a JMS error.
- InvalidDestinationException if the topic specified is not valid.

See TopicSession.unsubscribe
createDurableSubscriber

```java
public TopicSubscriber createDurableSubscriber(Topic topic, java.lang.String name, java.lang.String messageSelector, boolean noLocal) throws JMSException
```

Create a durable subscriber to the specified topic. A client can change an existing durable subscription by creating a durable subscriber with the same name and a new topic or message selector or both.

---

**WebSphere MQ Event Broker note**

This method throws a JMSException when you have a direct connection to WebSphere MQ Event Broker.

**Parameters:**
- topic: the topic to subscribe to.
- name: the name used to identify this subscription.
- messageSelector: deliver only messages with properties that match the message selector expression. This value can be null.
- noLocal: if set, inhibits the delivery of messages published by its own connection.

**Throws:**
- JMSEception if a session fails to create a subscriber because of a JMS error or non valid selector.
- InvalidDestinationException if the topic specified is not valid.
- InvalidSelectorException if the message selector is not valid.

createPublisher

```java
public TopicPublisher createPublisher(Topic topic) throws JMSException
```

Create a publisher for the specified topic.

**Parameters:**
- topic: the topic to publish to, or null if this is an unidentified producer.

**Throws:**
- JMSEception if a session fails to create a publisher because of a JMS error.
- InvalidDestinationException if the topic specified is not valid.

createSubscriber

```java
public TopicSubscriber createSubscriber(Topic topic) throws JMSException
```

Create a non-durable subscriber to the specified topic.

**Parameters:**
- topic: the topic to subscribe to

**Throws:**
- JMSEception - if a session fails to create a subscriber because of a JMS error.
TopicSession

- InvalidDestinationException if the topic specified is not valid.

createSubscriber

public TopicSubscriber createSubscriber(Topic topic, java.lang.String messageSelector, boolean noLocal) throws JMSException

Create a non-durable subscriber to the specified topic.

Parameters:
- topic: the topic to subscribe to.
- messageSelector: deliver only messages with properties that match the message selector expression. This value can be null.
- noLocal: if set, inhibits the delivery of messages published by its own connection.

Throws:
- JMSException if a session fails to create a subscriber because of a JMS error or non valid selector.
- InvalidDestinationException if the topic specified is not valid.
- InvalidSelectorException if the message selector is not valid.

createTemporaryTopic

public TemporaryTopic createTemporaryTopic() throws JMSException

Create a temporary topic. Its lifetime is that of the TopicConnection unless deleted earlier.

Returns:
A temporary topic.

Throws:
JMSException if a session fails to create a temporary topic because of a JMS error.

createTopic

public Topic createTopic(java.lang.String topicName) throws JMSException

Create a topic given a URI format topic name. (URI format is described in “Creating topics at runtime” on page 223. For information specific to JMS 1.1, see “Destinations” on page 239.) This allows you to create a topic with a provider-specific name.

Note: Clients that depend on this ability are not portable.

Parameters:
- topicName: the name of this topic.

Returns:
A topic with the given name.

Throws:
JMSException if a session fails to create a topic because of a JMS error.
unsubscribe

```java
public void unsubscribe(java.lang.String name)
    throws JMSException
```

Unsubscribe a durable subscription that has been created by a client.

---

**WebSphere MQ Event Broker note**

This method throws a JMSException when you have a direct connection to WebSphere MQ Event Broker.

**Note:** Do not use this method while an active subscription exists. You must close() your subscriber first.

**Parameters:**

- name: the name used to identify this subscription.

**Throws:**

- JMSException if JMS fails to unsubscribe the durable subscription because of a JMS error.
- InvalidDestinationException if the subscription name specified is not valid.
public interface TopicSubscriber extends MessageConsumer

WebSphere MQ class: MQTopicSubscriber

java.lang.Object
   +----com.ibm.mq.jms.MQMessageConsumer
      +----com.ibm.mq.jms.MQTopicSubscriber

A client uses a TopicSubscriber to receive messages that have been published to a topic. TopicSubscriber is the publish/subscribe variant of a JMS message consumer.

See also: MessageConsumer and TopicSession.createSubscriber

MQTopicSubscriber inherits the following methods from MQMessageConsumer:
   close
   getMessageListener
   receive
   receiveNoWait
   setMessageListener

Methods

getNoLocal
   public boolean getNoLocal() throws JMSException

   Get the NoLocal attribute for this TopicSubscriber. The default value for this attribute is false.

   Returns:
      Set to true if locally-published messages are being inhibited.

   Throws:
      JMSException if JMS fails to get NoLocal attribute for this topic subscriber because of an internal error.

getTopic
   public Topic getTopic() throws JMSException

   Get the topic associated with this subscriber.

   Returns:
      This subscriber's topic.

   Throws:
      JMSException if JMS fails to get topic for this topic subscriber because of an internal error.
public interface XAConnection
extends Connection
Subinterfaces: XAQueueConnection and XATopicConnection

WebSphere MQ class: MQXAConnection

java.lang.Object
| +----com.ibm.mq.jms.MQXAConnection

WebSphere MQ JMS exposes its JTS support in the XAConnectionFactory, XAConnection, and XASession classes. These classes are provided for use in a J2EE application server environment.

WebSphere Application Server Version 5 uses these classes to create and manage a pool of XAConnection and XASession objects. A JMS application does not need to use these classes directly if it is running in this environment.

A JMS application might need to use the XAConnection class only if it is running in a WebSphere Application Server environment with a version of WebSphere Application Server before Version 5. For more details, see Appendix E, “JMS JTA/XA interface with WebSphere Application Server V4,” on page 475.

See also: XAQueueConnection and XATopicConnection

Methods

createSession (JMS 1.1 only)

public Session createSession(boolean transacted,
   int acknowledgeMode) throws JMSException

Create a session.

Specified by:
createSession in the Connection interface.

Parameters:
• transacted: usage is undefined.
• acknowledgeMode: usage is undefined.

Returns:
A newly created session.

Throws:
JMSException if the XA connection fails to create a session because of an internal JMS error.
createXASession (JMS 1.1 only)

```java
public XASession createXASession() throws JMSException
```

Create an XA session.

**Returns:**

A newly created XA session.

**Throws:**

JMSException if the XA connection fails to create an XA session because of an internal JMS error.
public interface XAConnectionFactory
Subinterfaces: XAQueueConnectionFactory and XATopicConnectionFactory

WebSphere MQ class: MQXAConnectionFactory

WebSphere MQ JMS exposes its JTS support in the XAConnectionFactory, XAConnection, and XASession classes. These classes are provided for use in a J2EE application server environment.

WebSphere Application Server Version 5 uses these classes to create and manage a pool of XAConnection and XASession objects. A JMS application does not need to use these classes directly if it is running in this environment.

A JMS application might need to use the XAConnectionFactory class only if it is running in a WebSphere Application Server environment with a version of WebSphere Application Server before Version 5. For more details, see Appendix E, “JMS JTA/XA interface with WebSphere Application Server V4,” on page 475.

See also: XAQueueConnectionFactory and XATopicConnectionFactory

Methods

createXAConnection (JMS 1.1 only)

public XAConnection createXAConnection() throws JMSException

Create an XA connection with the default user identity. The connection is created in stopped mode. No messages are delivered until the Connection.start() method is called explicitly.

Returns:
A newly created XA connection.

Throws:
- JMSException if JMS fails to create an XA connection because of an internal JMS error.
- JMSSecurityException if client authentication fails because the user name or password is not valid.

createXAConnection (JMS 1.1 only)

public XAConnection createXAConnection(java.lang.String userName,
java.lang.String password)
throws JMSException

Create an XA connection with the specified user identity. The connection is created in stopped mode. No messages are delivered until the Connection.start method is called explicitly.

Parameters:
- userName: the user name of the caller.
- password: the password of the caller.
XAConnectionFactory

Returns:
A newly created XA connection.

Throws:
• JMSException if JMS fails to create an XA connection because of an internal JMS error.
• JMSSecurityException if client authentication fails because the user name or password is not valid.
public interface XAQueueConnection extends QueueConnection and XAConnection

WebSphere MQ class: MQXAQueueConnection

XAQueueConnection provides the same create options as QueueConnection. The only difference is that, by definition, an XAConnection is transacted. Refer to Appendix E, “JMS JTA/XA interface with WebSphere Application Server V4,” on page 475 for details about how WebSphere MQ JMS uses XA classes.

See also: XAConnection and QueueConnection

Methods

createQueueSession

public QueueSession createQueueSession(boolean transacted, int acknowledgeMode) throws JMSException

Create a QueueSession.

Parameters:

- transacted: if true, the session is transacted.
- acknowledgeMode: indicates whether the consumer or the client acknowledges any messages it receives. Possible values are:
  - Session.AUTO_ACKNOWLEDGE
  - Session.CLIENT_ACKNOWLEDGE
  - Session.DUPS_OK_ACKNOWLEDGE

This parameter is ignored if the session is transacted.

Returns:

A newly-created queue session (this is not an XA queue session).

Throws:

- JMSException if JMS Connection fails to create a queue session because of an internal error.

createXAQueueSession

public XAQueueSession createXAQueueSession()

Create an XAQueueSession.

Throws:

- JMSException if JMS Connection fails to create an XA queue session because of an internal error.
XAQueueConnectionFactory

public interface XAQueueConnectionFactory
extends QueueConnectionFactory and XAConnectionFactory

WebSphere MQ class: MQXAQueueConnectionFactory

java.lang.Object
+----com.ibm.mq.jms.MQConnectionFactory
| +----com.ibm.mq.jms.MQQueueConnectionFactory
|     +----com.ibm.mq.jms.MQXAQueueConnectionFactory

An XAQueueConnectionFactory provides the same create options as a QueueConnectionFactory. Refer to Appendix E, “JMS JTA/XA interface with WebSphere Application Server V4,” on page 475 for details about how WebSphere MQ JMS uses XA classes.

See also: QueueConnectionFactory and XAConnectionFactory

Methods

createXAQueueConnection

public XAQueueConnection createXAQueueConnection()
    throws JMSException

Create an XAQueueConnection using the default user identity. The connection is created in stopped mode. No messages are delivered until the Connection.start method is called explicitly.

Returns:
A newly-created XA queue connection.

Throws:
• JMSException if the JMS provider fails to create an XA queue connection because of an internal error.
• JMSSecurityException if client authentication fails because of a non valid user name or password.

createXAQueueConnection

public XAQueueConnection createXAQueueConnection(
    java.lang.String userName,
    java.lang.String password)
        throws JMSException

Create an XA queue connection using a specific user identity. The connection is created in stopped mode. No messages are delivered until the Connection.start method is called explicitly.

Parameters:
• userName: the user name of the caller.
• password: the password for the caller.

Returns:
A newly-created XA queue connection.
Throws:

- JMSException if the JMS Provider fails to create an XA queue connection because of an internal error.
- JMSSecurityException if client authentication fails because of a non valid user name or password.
public interface XAQueueSession
    extends XASession

WebSphere MQ class: MQXAQueueSession

java.lang.Object
    +----com.ibm.mq.jms.MQASession
        +----com.ibm.mq.jms.MQXAQueueSession

An XAQueueSession provides a regular QueueSession that can be used to create QueueReceivers, QueueSenders, and QueueBrowsers. Refer to Appendix E, “JMS JTA/XA interface with WebSphere Application Server V4,” on page 475 for details about how WebSphere MQ JMS uses XA classes.

The XAResource that corresponds to the QueueSession can be obtained by calling the getXAResource method, which is inherited from XASession.

See also: XASession

Methods

getQueueSession
    public QueueSession getQueueSession()
        throws JMSException

Get the queue session associated with this XAQueueSession.

Returns:
    The queue session object.

Throws:
    JMSException if a JMS error occurs.
XASession

public interface XASession
  extends java.langRunnable and Session

Subinterfaces: XAQueueSession and XATopicSession

WebSphere MQ class: MQXASession

java.lang.Object
  +----com.ibm.mq.jms.MQXASession

WebSphere MQ JMS exposes its JTS support in the XAConnectionFactory, XAConnection, and XASession classes. These classes are provided for use in a J2EE application server environment.

WebSphere Application Server Version 5 uses these classes to create and manage a pool of XAConnection and XASession objects. A JMS application does not need to use these classes directly if it is running in this environment.

A JMS application might need to use the XASession class only if it is running in a WebSphere Application Server environment with a version of WebSphere Application Server before Version 5. For more details, see Appendix E, “JMS JTA/XA interface with WebSphere Application Server V4,” on page 475.

See also: Session

Methods

commit

public void commit() throws JMSException

Do not call this method for an XASession object. If it is called, it throws a TransactionInProgressException.

Specified by:
commit in the Session interface.

Throws:
TransactionInProgressException if this method is called on an XASession.

gSession (JMS 1.1 only)

public Session gSession() throws JMSException

Get the session associated with thisXA session.

Returns:
The session.

Throws:
JMSException if JMS fails to return the session because of an internal JMS error.
XASession

getTransacted

```java
public boolean getTransacted() throws JMSException
```

Indicates whether the session is in transacted mode.

**Specified by:**
getTransacted in the Session interface.

**Returns:**
True.

**Throws:**
JMSException if JMS fails to return the transaction mode because of an internal JMS error.

getXAResource

```java
public javax.transaction.xa.XAResource getXAResource()
```

Return an XA resource to the caller.

**Returns:**
An XA resource to the caller.

rollback

```java
public void rollback() throws JMSException
```

Do not call this method for an XASession object. If it is called, it throws a TransactionInProgressException.

**Specified by:**
rollback in the Session interface.

**Throws:**
TransactionInProgressException if this method is called on an XASession.
public interface XATopicConnection extends TopicConnection and XAConnection

WebSphere MQ class: MQXATopicConnection

An XATopicConnection provides the same create options as TopicConnection. The only difference is that an XAConnection is transacted. Refer to Appendix E, “JMS JTA/XA interface with WebSphere Application Server V4,” on page 475 for details about how WebSphere MQ JMS uses XA classes.

See also: TopicConnection and XAConnection

Methods

createTopicSession

public TopicSession createTopicSession(boolean transacted,
                                      int acknowledgeMode)
                   throws JMSException

Create a TopicSession.

Specified by:
createTopicSession in interface TopicConnection.

Parameters:

• transacted: if true, the session is transacted.
• acknowledgeMode: one of:
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

Indicates whether the consumer or the client acknowledges any messages it receives. This parameter is ignored if the session is transacted.

Returns:

A newly-created topic session (this is not an XA topic session).

Throws:

JMSException if JMS Connection fails to create a topic session because of an internal error.

createXATopicSession

public XATopicSession createXATopicSession() throws JMSException

Create an XATopicSession.

Throws:

JMSException if the JMS Connection fails to create an XA topic session because of an internal error.
XATopicConnectionFactory

public interface XATopicConnectionFactory extends TopicConnectionFactory and XAConnectionFactory

WebSphere MQ class: MQXATopicConnectionFactory

java.lang.Object
     +----com.ibm.mq.jms.MQConnectionFactory
     |    +----com.ibm.mq.jms.MQTopicConnectionFactory
     |         +----com.ibm.mq.jms.MQXATopicConnectionFactory

An XATopicConnectionFactory provides the same create options as TopicConnectionFactory. Refer to Appendix E, "JMS JTA/XA interface with WebSphere Application Server V4," on page 475 for details about how WebSphere MQ JMS uses XA classes.

See also: TopicConnectionFactory and XAConnectionFactory

Methods

createXATopicConnection
    public XATopicConnection createXATopicConnection() throws JMSException

Create an XA topic connection using the default user identity. The connection is created in stopped mode. No messages are delivered until the Connection.start method is called explicitly.

Returns:
    A newly-created XA topic connection.

Throws:
    • JMSEception if the JMS Provider fails to create an XA topic connection because of an internal error.
    • JMSSecurityException if client authentication fails because of a non valid user name or password.

createXATopicConnection
    public XATopicConnection createXATopicConnection(java.lang.String userName, java.lang.String password) throws JMSEception

Create an XA topic connection using the specified user identity. The connection is created in stopped mode. No messages are delivered until the Connection.start method is called explicitly.

Parameters:
    • userName: the user name of the caller
    • password: the password of the caller

Returns:
    A newly-created XA topic connection.

Throws:
• JMSException if the JMS Provider fails to create an XA topic connection because of an internal error.
• JMSSecurityException if client authentication fails because of a non valid user name or password.
XATopicSession

public interface XATopicSession
extends XASession

WebSphere MQ class: MQXATopicSession

java.lang.Object
  +----com.ibm.mq.jms.MQSession
  |     +----com.ibm.mq.jms.MQXASession
  |     |     +----com.ibm.mq.jms.MQXATopicSession

An XATopicSession provides a TopicSession, which you can use to create TopicSubscribers and TopicPublishers. Refer to [Appendix E, “JMS JTA/XA interface with WebSphere Application Server V4,” on page 475](#) for details about how WebSphere MQ JMS uses XA classes.

The XAResource that corresponds to the TopicSession can be obtained by calling the getXAResource method, which is inherited from XASession.

See also: TopicSession and XASession

Methods

getTopicSession

    public TopicSession getTopicSession()
    throws JMSException

Get the topic session associated with this XATopicSession.

Returns:
    The topic session object.

Throws:
    • JMSException if a JMS error occurs.
Part 4. Appendixes
Appendix A. Mapping between administration tool properties and programmable properties

WebSphere MQ classes for Java Message Service provides facilities to set and query the properties of administered objects either using the WebSphere MQ JMS administration tool, or in an application program. Table 38 shows the mapping between each property name used with the administration tool and the corresponding member variable it refers to. It also shows the mapping between symbolic property values used in the tool and their programmable equivalents.

Table 38. Comparison of representations of property values within the administration tool and within programs

<table>
<thead>
<tr>
<th>Property</th>
<th>Member variable name</th>
<th>Tool property values</th>
<th>Program property values</th>
</tr>
</thead>
<tbody>
<tr>
<td>BROKERCCDSUBQ</td>
<td>brokerCCDurSubQueue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BROKERCCSUBQ</td>
<td>brokerCCSubQueue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BROKERCONQ</td>
<td>brokerControlQueue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BROKERDURSUBQ</td>
<td>brokerDurSubQueue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BROKERPUBQ</td>
<td>brokerPubQueue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BROKERQMGR</td>
<td>brokerQueueManager</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BROKERSUBQ</td>
<td>brokerSubQueue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BROKERVER</td>
<td>brokerVersion</td>
<td>V1, V2</td>
<td>JMSC.MQJMS_BROKER_V1, JMSC.MQJMS_BROKER_V2</td>
</tr>
<tr>
<td>CCSID</td>
<td>CCSID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHANNEL</td>
<td>channel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLEANUP</td>
<td>cleanupLevel</td>
<td>NONE, SAFE, STRONG, ASPROP</td>
<td>JMSC.MQJMS_CLEANUP_NONE, JMSC.MQJMS_CLEANUP_SAFE, JMSC.MQJMS_CLEANUP_STRONG, JMSC.MQJMS_CLEANUP_AS_PROPERTY</td>
</tr>
<tr>
<td>CLEANUPINT</td>
<td>cleanupInterval</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLIENTID</td>
<td>clientId</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIRECTAUTH</td>
<td>directAuth</td>
<td>BASIC, CERTIFICATE</td>
<td>JMSC.MQJMS_DIRECTAUTH_BASIC, JMSC.MQJMS_DIRECTAUTH_CERTIFICATE</td>
</tr>
<tr>
<td>ENCODING</td>
<td>encoding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXPIRY</td>
<td>expiry</td>
<td>APP, UNLIM</td>
<td>JMSC.MQJMS_EXP_APP, JMSC.MQJMS_EXP_UNLIMITED</td>
</tr>
<tr>
<td>FAILIFQUIESCE</td>
<td>failIfQuiesce</td>
<td>YES, NO</td>
<td>JMSC.MQJMS_FIQ_YES, JMSC.MQJMS_FIQ_NO</td>
</tr>
<tr>
<td>HOSTNAME</td>
<td>hostName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOCALADDRESS</td>
<td>localAddress</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSGBATCHSZ</td>
<td>msgBatchSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSGRETENTION</td>
<td>messageRetention</td>
<td>YES, NO</td>
<td>JMSC.MQJMS_MRET_YES, JMSC.MQJMS_MRET_NO</td>
</tr>
<tr>
<td>MSGSELECTION</td>
<td>messageSelection</td>
<td>CLIENT, BROKER</td>
<td>JMSC.MQJMS_MSEL_CLIENT, JMSC.MQJMS_MSEL_BROKER</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 1997, 2004
### Properties

Table 38. Comparison of representations of property values within the administration tool and within programs (continued)

<table>
<thead>
<tr>
<th>Property</th>
<th>Member variable name</th>
<th>Tool property values</th>
<th>Program property values</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULTICAST</td>
<td>multicast</td>
<td>DISABLED</td>
<td>JMSC.MQJMS_MULTICAST_DISABLED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOTR</td>
<td>JMSC.MQJMS_MULTICAST_NOT_RELIABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RELIABLE</td>
<td>JMSC.MQJMS_MULTICAST_RELIABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ENABLED</td>
<td>JMSC.MQJMS_MULTICAST_ENABLED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASCF</td>
<td>JMSC.MQJMS_MULTICAST_AS_CF</td>
</tr>
<tr>
<td>PERSISTENCE</td>
<td>persistence</td>
<td>APP</td>
<td>JMSC.MQJMS_PER_APP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QDEF</td>
<td>JMSC.MQJMS_PER_QDEF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PERS</td>
<td>JMSC.MQJMS_PER_PER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NON</td>
<td>JMSC.MQJMS_PER_NON</td>
</tr>
<tr>
<td>POLLINGINT</td>
<td>pollingInterval</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PORT</td>
<td>port</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRIORITY</td>
<td>priority</td>
<td>APP</td>
<td>JMSC.MQJMS_PRI_APP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QDEF</td>
<td>JMSC.MQJMS_PRI_QDEF</td>
</tr>
<tr>
<td>PROXYHOSTNAME</td>
<td>proxyHostName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROXYPORT</td>
<td>proxyPort</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUBACKINT</td>
<td>pubAckInterval</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUEUE</td>
<td>baseQueueName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QMANAGER</td>
<td>queueManager</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECEXIT</td>
<td>receiveExit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECEXITINIT</td>
<td>receiveExitInit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECEXIT</td>
<td>securityExit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECEXITINIT</td>
<td>securityExitInit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SENDEDIT</td>
<td>sendExit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SENDEXITINIT</td>
<td>sendExitInit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPARSESUBS</td>
<td>sparseSubscriptions</td>
<td>YES</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NO</td>
<td>false</td>
</tr>
<tr>
<td>SSLCIPHERSUITE</td>
<td>sslCipherSuite</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSLCRL</td>
<td>sslCertStores</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSLPEERNAME</td>
<td>sslPeerName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATREFRESHINT</td>
<td>statusRefreshInterval</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBSTORE</td>
<td>subscriptionStore</td>
<td>MIGRATE</td>
<td>JMSC.MQJMS_SUBSTORE_MIGRATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QUEUE</td>
<td>JMSC.MQJMS_SUBSTORE_QUEUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BROKER</td>
<td>JMSC.MQJMS_SUBSTORE_BROKER</td>
</tr>
<tr>
<td>SYNCPOINTALLGETS</td>
<td>syncpointAllGets</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TARGCLIENT</td>
<td>targetClient</td>
<td>JMS</td>
<td>JMSC.MQJMS_CLIENT_JMS_COMPLIANT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MQ</td>
<td>JMSC.MQJMS_CLIENT_NONJMS_MQ</td>
</tr>
<tr>
<td>TEMPMODEL</td>
<td>temporaryModel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOPIC</td>
<td>baseTopicName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRANSPORT</td>
<td>transportType</td>
<td>BIND</td>
<td>JMSC.MQJMS_TP_BINDINGS_MQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLIENT</td>
<td>JMSC.MQJMS_TP_CLIENT_MQ_TCPPI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DIRECT</td>
<td>JMSC.MQJMS_TP_DIRECT_TCPPI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DIRECTHTTPS</td>
<td>JMSC.MQJMS_TP_DIRECT_HTTPS</td>
</tr>
<tr>
<td>USECONNPOLLING</td>
<td>useConnectionPooling</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 38. Comparison of representations of property values within the administration tool and within programs (continued)

<table>
<thead>
<tr>
<th>Property</th>
<th>Member variable name</th>
<th>Tool property values</th>
<th>Program property values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Note:</td>
<td>* for an MQQueue object, the member variable name is baseQueueManagerName</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Properties

Appendix A. Mapping between administration tool properties and programmable properties
Properties
Appendix B. Scripts provided with WebSphere MQ classes for Java Message Service

The following files are provided in the bin directory of your WebSphere MQ JMS installation. These scripts are provided to assist with common tasks that need to be performed while installing or using WebSphere MQ JMS. Table 39 lists the scripts and their uses.

Table 39. Utilities supplied with WebSphere MQ classes for Java Message Service

<table>
<thead>
<tr>
<th>Utility</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cleanup.bat</td>
<td>Runs the subscription cleanup utility as described in &quot;Manual cleanup&quot; on page 232, or the consumer cleanup utility as described in &quot;Manual cleanup&quot; on page 250.</td>
</tr>
<tr>
<td>DefaultConfiguration</td>
<td>Runs the default configuration application on non-Windows systems as described in &quot;JMS Postcard configuration&quot; on page 22.</td>
</tr>
<tr>
<td>formatLog.bat</td>
<td>Converts binary log files to plain text as described in &quot;Logging&quot; on page 39.</td>
</tr>
<tr>
<td>IVTRun.bat</td>
<td>Runs the point-to-point installation verification test program as described in &quot;Running the point-to-point IVT&quot; on page 31.</td>
</tr>
<tr>
<td>IVTTidy.bat</td>
<td></td>
</tr>
<tr>
<td>IVTSetup.bat</td>
<td></td>
</tr>
<tr>
<td>JMSAdmin.bat</td>
<td>Runs the administration tool as described in Chapter 5, &quot;Using the WebSphere MQ JMS administration tool,&quot; on page 41.</td>
</tr>
<tr>
<td>JMSAdmin.config</td>
<td>Configuration file for the administration tool as described in &quot;Configuration&quot; on page 42.</td>
</tr>
<tr>
<td>postcard.bat</td>
<td>Starts the JMS Postcard application as described in &quot;JMS Postcard&quot; on page 19.</td>
</tr>
<tr>
<td>PSIVTRun.bat</td>
<td>Runs the publish/subscribe installation verification test program as described in &quot;The publish/subscribe installation verification test&quot; on page 35.</td>
</tr>
<tr>
<td>PSReportDump.class</td>
<td>Views broker report messages as described in &quot;Handling broker reports&quot; on page 235. For information specific to JMS 1.1, see &quot;Handling broker reports&quot; on page 252.</td>
</tr>
<tr>
<td>runjms.bat</td>
<td>Helps you to run JMS applications as described in &quot;Running your own WebSphere MQ JMS programs&quot; on page 38.</td>
</tr>
</tbody>
</table>

Note: On UNIX systems, the extension .bat is omitted from the filenames.
Scripts
Appendix C. LDAP schema definition for storing Java objects

This appendix gives details of the schema definitions (objectClass and attribute definitions) needed in an LDAP directory for it to store Java objects. Read it if you want to use an LDAP server as your JNDI service provider in which to store WebSphere MQ JMS administered objects. Ensure that your LDAP server schema contains the following definitions; the exact procedure to achieve this varies from server to server. How to make the changes to some specific LDAP servers is covered later in this section.

Much of the data contained in this appendix has been taken from RFC 2713 Schema for Representing Java Objects in an LDAP Directory, which can be found at http://www.faqs.org/rfcs/rfc2713.html. LDAP server-specific information has been taken from Sun Microsystems' JNDI 1.2.1 LDAP service provider, available at http://java.sun.com/products/jndi.

Checking your LDAP server configuration

To check whether the LDAP server is already configured to accept Java objects, run the WebSphere MQ JMS administration tool JMSAdmin against your LDAP server (see “Invoking the administration tool” on page 41).

Try to create and display a test object using the following commands:

```bash
DEFINE QCF(ldapTest)
DISPLAY QCF(ldapTest)
```

If no errors occur, your server is properly configured to store Java objects and you can proceed to store JMS objects. However, if your LDAP server contains older schema definitions (for example, from an earlier draft of RFC 2713 such as the now-obsolete draft-ryan-java-schema-00 and draft-ryan-java-schema-01 specifications), update them with those described here.

If a SchemaViolationException occurs, or if the message Unable to bind to object is returned, your server is not properly configured. Either your server is not configured to store Java objects, permissions on the objects are not correct, or the provided suffix or context has not been set up. The following information helps you with the schema configuration part of your server setup.
### Table 40. Attribute settings for javaCodebase

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID (Object Identifier)</td>
<td>1.3.6.1.4.1.42.2.27.4.1.7</td>
</tr>
<tr>
<td>Syntax</td>
<td>IA5 String (1.3.6.1.4.1.1466.115.121.1.26)</td>
</tr>
<tr>
<td>Maximum length</td>
<td>2048</td>
</tr>
<tr>
<td>Single/multi-valued</td>
<td>Multi-valued</td>
</tr>
<tr>
<td>User modifiable</td>
<td>Yes</td>
</tr>
<tr>
<td>Matching rules</td>
<td>caseExactIA5Match</td>
</tr>
<tr>
<td>Access class</td>
<td>normal</td>
</tr>
<tr>
<td>Usage</td>
<td>userApplications</td>
</tr>
<tr>
<td>Description</td>
<td>URL(s) specifying the location of class definition</td>
</tr>
</tbody>
</table>

### Table 41. Attribute settings for javaClassName

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID (Object Identifier)</td>
<td>1.3.6.1.4.1.42.2.27.4.1.6</td>
</tr>
<tr>
<td>Syntax</td>
<td>Directory String (1.3.6.1.4.1.1466.115.121.1.15)</td>
</tr>
<tr>
<td>Maximum length</td>
<td>2048</td>
</tr>
<tr>
<td>Single/multi-valued</td>
<td>Single-valued</td>
</tr>
<tr>
<td>User modifiable?</td>
<td>Yes</td>
</tr>
<tr>
<td>Matching rules</td>
<td>caseExactMatch</td>
</tr>
<tr>
<td>Access class</td>
<td>normal</td>
</tr>
<tr>
<td>Usage</td>
<td>userApplications</td>
</tr>
<tr>
<td>Description</td>
<td>Fully qualified name of distinguished Java class or interface</td>
</tr>
</tbody>
</table>

### Table 42. Attribute settings for javaClassNames

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID (Object Identifier)</td>
<td>1.3.6.1.4.1.42.2.27.4.1.13</td>
</tr>
<tr>
<td>Syntax</td>
<td>Directory String (1.3.6.1.4.1.1466.115.121.1.15)</td>
</tr>
<tr>
<td>Maximum length</td>
<td>2048</td>
</tr>
<tr>
<td>Single/multi-valued</td>
<td>Multi-valued</td>
</tr>
<tr>
<td>User modifiable</td>
<td>Yes</td>
</tr>
<tr>
<td>Matching rules</td>
<td>caseExactMatch</td>
</tr>
<tr>
<td>Access class</td>
<td>normal</td>
</tr>
<tr>
<td>Usage</td>
<td>userApplications</td>
</tr>
<tr>
<td>Description</td>
<td>Fully qualified Java class or interface name</td>
</tr>
</tbody>
</table>
### Table 43. Attribute settings for javaFactory

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID (Object Identifier)</td>
<td>1.3.6.1.4.1.42.2.27.4.1.10</td>
</tr>
<tr>
<td>Syntax</td>
<td>Directory String (1.3.6.1.4.1.1466.115.121.1.15)</td>
</tr>
<tr>
<td>Maximum length</td>
<td>2048</td>
</tr>
<tr>
<td>Single/multi-valued</td>
<td>Single-valued</td>
</tr>
<tr>
<td>User modifiable</td>
<td>Yes</td>
</tr>
<tr>
<td>Matching rules</td>
<td>caseExactMatch</td>
</tr>
<tr>
<td>Access class</td>
<td>normal</td>
</tr>
<tr>
<td>Usage</td>
<td>userApplications</td>
</tr>
<tr>
<td>Description</td>
<td>Fully qualified Java class name of a JNDI object factory</td>
</tr>
</tbody>
</table>

### Table 44. Attribute settings for javaReferenceAddress

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID (Object Identifier)</td>
<td>1.3.6.1.4.1.42.2.27.4.1.11</td>
</tr>
<tr>
<td>Syntax</td>
<td>Directory String (1.3.6.1.4.1.1466.115.121.1.15)</td>
</tr>
<tr>
<td>Maximum length</td>
<td>2048</td>
</tr>
<tr>
<td>Single/multi-valued</td>
<td>Multi-valued</td>
</tr>
<tr>
<td>User modifiable</td>
<td>Yes</td>
</tr>
<tr>
<td>Matching rules</td>
<td>caseExactMatch</td>
</tr>
<tr>
<td>Access class</td>
<td>normal</td>
</tr>
<tr>
<td>Usage</td>
<td>userApplications</td>
</tr>
<tr>
<td>Description</td>
<td>Addresses associated with a JNDI Reference</td>
</tr>
</tbody>
</table>

### Table 45. Attribute settings for javaSerializedData

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID (Object Identifier)</td>
<td>1.3.6.1.4.1.42.2.27.4.1.8</td>
</tr>
<tr>
<td>Syntax</td>
<td>Octet String (1.3.6.1.4.1.1466.115.121.1.40)</td>
</tr>
<tr>
<td>Single/multi-valued</td>
<td>Single-valued</td>
</tr>
<tr>
<td>User modifiable</td>
<td>Yes</td>
</tr>
<tr>
<td>Access class</td>
<td>normal</td>
</tr>
<tr>
<td>Usage</td>
<td>userApplications</td>
</tr>
<tr>
<td>Description</td>
<td>Serialized form of a Java object</td>
</tr>
</tbody>
</table>

### objectClass definitions

### Table 46. objectClass definition for javaSerializedObject

<table>
<thead>
<tr>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID (Object Identifier)</td>
<td>1.3.6.1.4.1.42.2.27.4.2.5</td>
</tr>
<tr>
<td>Extends/superior</td>
<td>javaObject</td>
</tr>
<tr>
<td>Type</td>
<td>AUXILIARY</td>
</tr>
<tr>
<td>Required (must) attrs</td>
<td>javaSerializedData</td>
</tr>
</tbody>
</table>
objectClass definitions

Table 47. objectClass definition for javaObject

<table>
<thead>
<tr>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID (Object Identifier)</td>
<td>1.3.6.1.4.1.42.2.27.4.2.4</td>
</tr>
<tr>
<td>Extends/superior</td>
<td>top</td>
</tr>
<tr>
<td>Type</td>
<td>ABSTRACT</td>
</tr>
<tr>
<td>Required (must) attrs</td>
<td>javaClassName</td>
</tr>
<tr>
<td>Optional (may) attrs</td>
<td>javaClassNames javaCodebase javaDoc description</td>
</tr>
</tbody>
</table>

Table 48. objectClass definition for javaContainer

<table>
<thead>
<tr>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID (Object Identifier)</td>
<td>1.3.6.1.4.1.42.2.27.4.2.1</td>
</tr>
<tr>
<td>Extends/superior</td>
<td>top</td>
</tr>
<tr>
<td>Type</td>
<td>STRUCTURAL</td>
</tr>
<tr>
<td>Required (must) attrs</td>
<td>cn</td>
</tr>
</tbody>
</table>

Table 49. objectClass definition for javaNamingReference

<table>
<thead>
<tr>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID (Object Identifier)</td>
<td>1.3.6.1.4.1.42.2.27.4.2.7</td>
</tr>
<tr>
<td>Extends/superior</td>
<td>javaObject</td>
</tr>
<tr>
<td>Type</td>
<td>AUXILIARY</td>
</tr>
<tr>
<td>Optional (may) attrs</td>
<td>javaReferenceAddress javaFactory</td>
</tr>
</tbody>
</table>

Server-specific configuration details

This section describes special steps you need to take to use the following servers:

- "Netscape Directory (4.1 and earlier)"
- "Microsoft Active Directory"
- "Sun Microsystems’ schema modification applications" on page 467
- "OS/400 V4R5 Schema Modification" on page 467

Netscape Directory (4.1 and earlier)

This level of Netscape Directory does not support the Octet String syntax; use Binary syntax (1.3.6.1.4.1.1466.115.121.1.5) instead. Netscape Directory 4.1 also has problems parsing an object class definition that contains a MUST clause without parentheses. The workaround is to add a superfluous value (objectClass) to each MUST clause.

Alternatively, you can use the Sun-supplied schema modification applications described in “Sun Microsystems’ schema modification applications” on page 467

Microsoft Active Directory

Within Active Directory, only the names of structural classes (not auxiliary classes) can appear in the object class attribute of an entry. You must redefine the abstract and auxiliary classes in the Java schema definition as structural. This has the following effects:

- The javaObject class now inherits from javaContainer
• The javaNamingReference and javaSerializedObject classes now inherit from javaObject

Instead of making these changes manually, you can use the Sun-supplied schema modification applications described in "Sun Microsystems’ schema modification applications."

Sun Microsystems’ schema modification applications
You can use your LDAP server’s administration tool (for example, the Directory Management Tool for IBM’s SecureWay® Directory) to verify or add the definitions described above. Alternatively, Sun Microsystems’ JNDI 1.2.1 LDAP service provider (available at http://java.sun.com/products/jndi) contains Java applications (CreateJavaSchema.java and UpdateJavaSchema.java) that add or update the required schema definitions automatically. These applications contain workarounds for schema bugs and server-specific behavior in both Netscape Directory Server (pre-4.1 and 4.1) and Microsoft® Windows 2000 Active Directory.

These applications are not packaged with WebSphere MQ classes for Java Message Service. Details on running them can be found in both the README and the application source contained in the Sun JNDI 1.2.1 LDAP service provider download.

OS/400 V4R5 Schema Modification
You can use your LDAP server’s administration tool (the Directory Management Tool for IBM’s SecureWay Directory) to verify or add the definitions described above.

OS/400 V4R5 LDAP Server is shipped with an out-of-date version of RFC 2713 schema for Java objects. Update this schema to the schema described above to operate correctly with JMSAdmin. When you modify the schema, delete any out-of-data definitions and uses of those definitions before adding the correct definitions.

OS/400 V5R1 is shipped with the current version of RFC 2713 and does not require these changes.

Microsoft Active Directory
Appendix D. Connecting to other products

This section covers:

- How to configure a publish/subscribe broker for a connection from WebSphere MQ JMS in “Setting up a publish/subscribe broker.”
- How to use WebSphere MQ Integrator V2 to route or transform messages sent to or from a JMS client in “Transformation and routing with WebSphere MQ Integrator V2” on page 471.
- How to configure WebSphere MQ JMS for a direct connection to WebSphere Business Integration Event Broker Version 5.0 or WebSphere Business Integration Message Broker Version 5.0 in “Configuring WebSphere MQ JMS for a direct connection to WebSphere Business Integration Event Broker Version 5.0 and WebSphere Business Integration Message Broker Version 5.0” on page 472.

Setting up a publish/subscribe broker

You can use WebSphere MQ Integrator Version 2, WebSphere MQ Event Broker Version 2.1, WebSphere Business Integration Event Broker Version 5.0, or WebSphere Business Integration Message Broker Version 5.0 as the publish/subscribe broker for WebSphere MQ JMS. You can link to each of these brokers across a connection to base WebSphere MQ, or you can connect directly to WebSphere MQ Event Broker, WebSphere Business Integration Event Broker, or WebSphere Business Integration Message Broker over TCP/IP. Each method requires some setup activities:

Linking across WebSphere MQ

- Base WebSphere MQ
  
  First, create a broker publication queue. This is a WebSphere MQ queue on the broker queue manager; it is used to submit publications to the broker. You can choose your own name for this queue, but it must match the queue name in your TopicConnectionFactory’s BROKERPUBQ property. By default, a TopicConnectionFactory’s BROKERPUBQ property is set to the value SYSTEM.BROKER.DEFAULT.STREAM so, unless you want to configure a different name in the TopicConnectionFactory, name the queue SYSTEM.BROKER.DEFAULT.STREAM.

- WebSphere MQ Integrator V2
  
  The next step is to set up a message flow within an execution group for the broker. The purpose of this message flow is to read messages from the broker publication queue. (If you want, you can set up multiple publication queues; each needs its own TopicConnectionFactory and message flow.)

  The basic message flow consists of an MQInput node (configured to read from the SYSTEM.BROKER.DEFAULT.STREAM queue) whose output is connected to the input of a Publication (or MQOutput) node.

  The message flow diagram therefore looks similar to the following:
When this message flow is deployed and the broker is started, from the JMS application’s perspective the WebSphere MQ Integrator V2 broker behaves like an MQSeries Publish/Subscribe broker. The current subscription state can be viewed using the WebSphere MQ Integrator Control Center.

Notes:
1. No modifications are required to WebSphere MQ classes for Java Message Service.
2. MQSeries Publish/Subscribe and WebSphere MQ Integrator V2 brokers cannot coexist on the same queue manager.
3. Details of the WebSphere MQ Integrator V2 installation and setup procedure are described in the WebSphere MQ Integrator for Windows NT Version 2.0 Installation Guide.

Direct connection to WebSphere MQ Event Broker Version 2.1 over TCP/IP
For this, set up a message flow within an execution group on WebSphere MQ Event Broker. This message flow is to read messages from the TCP/IP socket on which the broker is listening.

The basic message flow consists of a JMSIPOptimised flow set to listen on the port configured for direct connections. By default, this port is 1506.

Note: WebSphere MQ Event Broker can be configured to listen for both direct connections across TCP/IP from WebSphere MQ JMS and connections made across TCP/IP through WebSphere MQ. In this case, the two listeners must be configured on different ports. The default port for a WebSphere MQ connection is 1414.

Direct connection to WebSphere Business Integration Event Broker Version 5.0
or WebSphere Business Integration Message Broker Version 5.0
To configure a WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker broker for a direct connection from WebSphere MQ JMS, create a message flow to read messages from the TCP/IP port on which the broker is listening and publish the messages. You can do this in either of the following ways:

- You can create a message flow that contains a Real-timeOptimizedFlow message processing node.
- You can create a message flow that contains a Real-timeInput message processing node and a Publication message processing node.

You must configure the Real-timeOptimizedFlow or Real-timeInput node to listen on the port used for direct connections. By default, the port number for direct connections is 1506.
Transformation and routing with WebSphere MQ Integrator V2

You can use WebSphere MQ Integrator V2 to route or transform messages that are created by a JMS client application, and to send or publish messages to a JMS client.

The WebSphere MQ JMS implementation uses the mcd folder of the MQRFH2 to carry information about the message, as described in “The MQRFH2 header” on page 262. By default, the Message Domain (Msd) property is used to identify whether the message is a text, bytes, stream, map, or object message. This value is set depending on the type of the JMS message.

If the application calls setJMSType, it can set the mcd type field to a value of its choosing. This type field can be read by the WebSphere MQ Integrator message flow, and a receiving JMS application can use the getJMSType method to retrieve its value. This applies to all kinds of JMS message.

When a JMS application creates a text or bytes message, the application can set mcd folder fields explicitly by calling the setJMSType method and passing in a string argument in a special URI format as follows:

```
mcd://domain/[set]/[/type] [/?format=fmt]
```

This URI form allows an application to set the mcd to a domain that is not one of the standard jms_xxxx values; for example, to domain mrm. It also allows the application to set any or all of the mcd set, type, and format fields.

The string argument to setJMSType is interpreted as follows:

1. If the string does not appear to be in the special URI format (it does not start with mcd://), the string is added to the mcd folder as the type field.
2. If the string starts with mcd://, conforms to the URI format, and the message is a Text or Bytes message, the URI string is split into its constituent parts. The domain part overrides the jms_text or jms_bytes value that would otherwise have been generated, and the remaining parts (if present) are used to set the set, type, and format fields in the mcd. Note that set, type, and format are all optional.
3. If the string starts with mcd:// and the message is a Map, Stream, or Object message, the setJMSType call throws an exception. So you cannot override the domain, or provide a set or format for these classes of message, but you can provide a type.

When a WebSphere MQ message is received with an Msd domain other than one of the standard jms_xxxx values, it is instantiated as a JMS text or bytes message and a URI-style JMSType is assigned to it. If the format field of the RFH2 is MQFMT_STRING, it becomes a TextMessage; otherwise it becomes a BytesMessage. The receiving application can read this using the getJMSType method.
WebSphere Business Integration brokers

Configuring WebSphere MQ JMS for a direct connection to WebSphere Business Integration Event Broker Version 5.0 and WebSphere Business Integration Message Broker Version 5.0

A WebSphere MQ JMS client can connect directly to a WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker broker over TCP/IP. The available function is comparable to that provided for a direct connection to a WebSphere MQ Event Broker Version 2.1 broker, but with the following additions:

- Secure Sockets Layer (SSL) authentication
- Multicast
- HTTP tunnelling
- Connect via proxy

For detailed information about this additional function, see the WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker Information Center. The following sections explain how to configure a WebSphere MQ JMS client in order to use this function.

Secure Sockets Layer (SSL) authentication

You can use SSL authentication when a WebSphere MQ JMS client connects directly to a WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker broker. Only SSL authentication is supported for this type of connection. SSL cannot be used to encrypt or decrypt message data that flows between the WebSphere MQ JMS client and the broker or to perform integrity checks on the data.

Note the difference between this situation and that when a WebSphere MQ JMS client connects to a WebSphere MQ queue manager. In the latter case, the WebSphere MQ SSL support can be used to encrypt and decrypt message data that flows between the client and the queue manager and to perform integrity checks on the data, as well as providing authentication.

If you want to protect message data on a direct connection to a broker, you can use function in the broker instead. You can assign a quality of protection (QoP) value to each topic whose associated messages you want to protect. This allows you to select a different level of message protection for each topic.

If client authentication is required, a WebSphere MQ JMS client can use the same digital certificate for connecting directly to a broker as it does for connecting to a WebSphere MQ queue manager.

You can configure a WebSphere MQ JMS client to use SSL authentication in either of the following ways:

- In a WebSphere MQ JMS application, use the setDirectAuth() method of an MQConnectionFactory or MQTopicConnectionFactory object to set the direct authentication attribute to MQSC.MQJMS_DIRECTAUTH_CERTIFICATE.
- Use the WebSphere MQ JMS administration tool to set the DIRECTAUTH property to CERTIFICATE.

Notes:

1. If the TRANSPORT property is set to DIRECT, then it is the DIRECTAUTH property, not the SSLCIPHERSUITE property, that determines whether SSL authentication is used.
2. If the DIRECTAUTH property is set to CERTIFICATE, the SSLPEERNAME and SSLCRL properties are used to perform the same checks as those performed when a WebSphere MQ JMS client connects to a WebSphere MQ queue manager using the WebSphere MQ SSL support.

3. The Java Secure Socket Extension (JSSE) KeyStore and TrustStore configurations determine which client certificate is used for authentication, and whether a server certificate is trusted, in the same way that they do when a WebSphere MQ JMS client connects to a WebSphere MQ queue manager using the WebSphere MQ SSL support.

Multicast

You can configure a WebSphere MQ JMS client multicast connection to a broker in either of the following ways:

- In a WebSphere MQ JMS application, use the setMulticast() method of an MQConnectionFactory, MQTopicConnectionFactory, or MQTopic object to set the multicast attribute.
- Use the WebSphere MQ JMS administration tool to set the MULTICAST property.

The TRANSPORT property must be set to DIRECT before the MULTICAST property has any effect.

HTTP tunnelling

A WebSphere MQ JMS client can connect to a broker using HTTP tunnelling. HTTP tunnelling is suitable for applets because the Java 2 Security Manager normally rejects any attempt by an applet to connect directly to the broker. Using HTTP tunnelling, which exploits the built in support in Web browsers, a WebSphere MQ JMS client can connect to the broker using the HTTP protocol as though connecting to a Web site.

You can configure a WebSphere MQ JMS client to use HTTP tunnelling in either of the following ways:

- In a WebSphere MQ JMS application, use the setTransportType() method of an MQConnectionFactory object to set the transport type attribute to JMSC.MQJMS_TP_DIRECT_HTTP.
- Use the WebSphere MQ JMS administration tool to set the TRANSPORT property to DIRECTHTTP.

SSL authentication cannot be used with HTTP tunnelling.

Connect via proxy

A WebSphere MQ JMS client can connect to a broker through a proxy server. The client connects directly to the proxy server and uses the Internet protocol defined in RFC 2817 to ask the proxy server to forward the connection request to the broker. This option does not work for applets because the Java 2 Security Manager normally rejects any attempt by an applet to connect directly to a proxy server.

You can configure a WebSphere MQ JMS client to connect to a broker through a proxy server in either of the following ways:

- In a WebSphere MQ JMS application, use the setProxyHostName() and setProxyPort() methods of an MQConnectionFactory or MQTopicConnectionFactory object to set the proxy host name and proxy port attributes.
WebSphere Business Integration brokers

- Use the WebSphere MQ JMS administration tool to set the PROXYHOSTNAME and PROXYPORT properties.

If the TRANSPORT property is set to DIRECT, the type of connection to the broker depends on the PROXYHOSTNAME property according to the following rules:

- If the PROXYHOSTNAME property is set to the empty string, the WebSphere MQ JMS client connects directly to the broker using the HOSTNAME and PORT properties to locate the broker.

- If the PROXYHOSTNAME property is set to a value other than the empty string, the WebSphere MQ JMS client connects to the broker through the proxy server identified by the PROXYHOSTNAME and PROXYPORT properties.
Appendix E. JMS JTA/XA interface with WebSphere Application Server V4

WebSphere MQ classes for Java Message Service includes the JMS XA interfaces. These allow WebSphere MQ JMS to participate in a two-phase commit that is coordinated by a transaction manager that complies with the Java Transaction API (JTA).

This section describes how to use these features with the WebSphere Application Server, Advanced Edition, so that WebSphere Application Server can coordinate JMS send and receive operations, and database updates, in a global transaction.

Note: Before you use WebSphere MQ JMS and the XA classes with WebSphere Application Server, there might be additional installation or configuration steps. Refer to the Readme.txt file on the WebSphere MQ Using Java SupportPac Web page for the latest information (www.ibm.com/software/ts/mqseries/txppacs/ma88.html).

Using the JMS interface with WebSphere Application Server

This section provides guidance on using the JMS interface with the WebSphere Application Server Version 4, Advanced Edition.

You must already understand the basics of JMS programs, WebSphere MQ, and EJB beans. These details are in the JMS specification, the EJB V2 specification (both available from Sun), this manual, the samples provided with WebSphere MQ JMS, and other manuals for WebSphere MQ and WebSphere Application Server.

Administered objects

JMS uses administered objects to encapsulate vendor-specific information. This minimizes the impact of vendor-specific details on end-user applications. Administered objects are stored in a JNDI namespace, and can be retrieved and used in a portable manner without knowing the vendor-specific contents.

For standalone use, WebSphere MQ JMS provides the following classes:
- MQQueueConnectionFactory
- MQQueue
- MQTopicConnectionFactory
- MQTopic

WebSphere Application Server provides an additional pair of administered objects so that WebSphere MQ JMS can integrate with WebSphere Application Server:
- JMSWrapXAQueueConnectionFactory
- JMSWrapXATopicConnectionFactory

You use these objects in exactly the same way as the MQQueueConnectionFactory and MQTopicConnectionFactory. However, behind the scenes they use the XA versions of the JMS classes, and enlist the WebSphere MQ JMS XAResource in the WebSphere Application Server transaction.
Container-managed versus bean-managed transactions

Container-managed transactions are transactions in EJB beans that are demarcated automatically by the EJB container. Bean-managed transactions are transactions in EJB beans that are demarcated by the program (using the UserTransaction interface).

Two-phase commit versus one-phase optimization

The WebSphere Application Server coordinator invokes a true two-phase commit only if more than one XAResource is used in a particular transaction. Transactions that involve a single resource are committed using a one-phase optimization. This largely removes the need to use different ConnectionFactories for distributed and non-distributed transactions.

Defining administered objects

You can use the WebSphere MQ JMS administration tool to define the WebSphere Application Server-specific connection factories and store them in a JNDI namespace. The admin.config file in MQ_install_dir/bin must contain the following lines:

```java
INITIAL_CONTEXT_FACTORY=com.ibm.websphere.naming.WsnInitialContextFactory
PROVIDER_URL=iiop://hostname/
```

MQ_install_dir is the installation directory for WebSphere MQ JMS, and hostname is the name or IP address of the machine that is running WebSphere Application Server.

To access the com.ibm.ejs.ns.jndi.CNInitialContextFactory, you must add the file ejs.jar from the WebSphere Application Server lib directory to the CLASSPATH.

To create the new factories, use the define verb with the following two new types:

```java
def WSQCF(name) [properties]
def WSTCF(name) [properties]
```

These new types use the same properties as the equivalent QCF or TCF types, except that only the BIND transport type is allowed (and therefore, client properties cannot be configured). For details, see "Administering JMS objects" on page 45.

Retrieving administration objects

In an EJB bean, you retrieve the JMS-administered objects using the InitialContext.lookup() method, for example:

```java
InitialContext ic = new InitialContext();
TopicConnectionFactory tcf = (TopicConnectionFactory) ic.lookup("jms/Samples/TCF1");
```

The objects can be cast to, and used as, the generic JMS interfaces. Normally, there is no need to program to the WebSphere MQ specific classes in the application code.

Samples

There are three samples that illustrate the basics of using WebSphere MQ JMS with WebSphere Application Server Advanced Edition. These are in subdirectories of MQ_samples_dir/ws, where MQ_samples_dir is the samples directory for WebSphere MQ JMS. See Table 3 on page 10 to find where this is.
JTA/XA interface with WebSphere Application Server V4

- Sample1 demonstrates a simple put and get for a message in a queue by using container-managed transactions.
- Sample2 demonstrates a simple put and get for a message in a queue by using bean-managed transactions.
- Sample3 illustrates the use of the publish/subscribe API.

For details about how to build and deploy the EJB beans, refer to the WebSphere Application Server documentation.

The readme.txt files in each sample directory include example output from each EJB bean. The scripts provided assume that a default queue manager is available on the local machine. If your installation varies from the default, you can edit these scripts.

Sample1

Sample1EJB.java, in the sample1 directory, defines two methods that use JMS:
- putMessage() sends a TextMessage to a queue, and returns the MessageID of the sent message
- getMessage() reads the message with the specified MessageID back from the queue

Before you run the sample, you must store two administered objects in the WebSphere Application Server JNDI namespace:

**QCF1** a WebSphere Application Server-specific queue connection factory

**Q1** a queue

Both objects must be bound in the jms/Samples sub-context.

To set up the administered objects, you can either use the WebSphere MQ JMS administration tool and set them up manually, or you can use the script provided.

The WebSphere MQ JMS administration tool must be configured to access the WebSphere Application Server namespace. For details about how to configure the administration tool, refer to "Configuring for WebSphere Application Server V3.5" on page 44.

To set up the administered objects with typical default settings, you can enter the following command to run the script admin.scp:

```
JMSAdmin < admin.scp
```

The bean must be deployed with the getMessage and putMessage methods marked as TX_REQUIRED. This ensures that the container starts a transaction before entering each method, and commits the transaction when the method completes. Within the methods, you do not need any application code that relates to the transactional state. However, the message sent from putMessage occurs under syncpoint, and does not become available until the transaction is committed.

In the sample1 directory, there is a simple client program, Sample1Client.java, to call the EJB bean. There is also a script, runClient, to simplify running this program.

The client program (or script) takes a single parameter, which is used as the body of a TextMessage that is sent by the EJB bean putMessage method. The getMessage
JTA/XA interface with WebSphere Application Server V4

is called to read the message back off the queue and return the body to the client for display. The EJB bean sends progress messages to the standard output (stdout) of the application server, so you might want to monitor that output during the run.

If the application server is on a machine that is remote from the client, you might need to edit Sample1Client.java. If you do not use the defaults, you might need to edit the runClient script to match the local installation path and name of the deployed jar file.

Sample2

Sample2EJB.java, in the sample2 directory, performs the same task as sample1, and requires the same administered objects. Unlike sample1, sample2 uses bean-managed transactions to control the transactional boundaries.

If you have not already run sample1, ensure that you set up the administered objects QCF1 and Q1, as described in “Sample1” on page 477.

The putMessage methods and getMessage methods start by obtaining an instance of UserTransaction. They use this instance to create a transaction using the UserTransaction.begin() method. After that, the main body of the code is the same as sample1 until the end of each method. At the end of each method, the transaction is completed by the UserTransaction.commit() call.

In the sample2 directory, there is a simple client program, Sample2Client.java, to call the EJB bean. There is also a script, runClient, to simplify running this program. You can use these in the same way as described for “Sample1” on page 477.

Sample3

Sample3EJB.java, in the sample3 directory, demonstrates the use of the publish/subscribe API with WebSphere Application Server. Publishing a message is very similar to the point-to-point case. However, there are differences when receiving messages using a TopicSubscriber.

Publish/subscribe programs commonly use nondurable subscribers. These nondurable subscribers exist only for the lifetime of their owning sessions (or less if the subscriber is closed explicitly). Also, they can receive messages from the broker only during that lifetime.

To convert sample1 to publish/subscribe, create a durable subscriber before the message is published. Durable subscribers persist as a deliverable end-point beyond the lifetime of the session. Therefore, the message is available for retrieval during the call to getMessage().

The EJB bean includes two additional methods:
• createSubscription creates a durable subscription
• destroySubscription deletes a durable subscription
These methods (along with putMessage and getMessage) must be deployed with the TX_REQUIRED attribute.
Before you run sample3, you must store two administered objects in the WebSphere Application Server JNDI namespace:

  TCF1
  T1

Both objects must be bound in the jms/Samples sub-context.

To set up the administered objects, you can either use the WebSphere MQ JMS administration tool and set them up manually, or you can use a script. The script admin.scp is provided in the sample3 directory.

The WebSphere MQ JMS administration tool must be configured to access the WebSphere Application Server namespace. For details about how to configure the administration tool, refer to “Configuring for WebSphere Application Server V3.5” on page 44.

To set up the administered objects with typical default settings, you can enter the following command to run the script admin.scp:

```
JMSAdmin < admin.scp
```

If you have already run admin.scp to set up objects for sample1 or sample2, there will be error messages when you run admin.scp for sample3. (These occur when you attempt to create the jms and Samples sub-contexts.) You can safely ignore these error messages.

Also, before you run sample3, ensure that the WebSphere MQ publish/subscribe broker (SupportPac MA0C) is installed and running.

In the sample3 directory, there is a simple client program, Sample3Client.java, to call the EJB bean. There is also a script, runClient, to simplify running this program. You can use these in the same way as described for “Sample1” on page 477.
Appendix F. Using WebSphere MQ Java in applets with Java 1.2 or later

You might need to perform additional tasks to run an applet using WebSphere MQ Java classes in a Java virtual machine (JVM) at Java 1.2 level or greater. This is because the default security rules for applets with JVMs at these levels were changed to reduce the risk of damage by malevolent or misbehaving classes.

There are two different approaches that you can take:
1. Change the security settings on the browser and JVM to allow the use of WebSphere MQ Java packages.
2. Copy the WebSphere MQ Java classes to the same location as the applet you wish to run.

Changing browser security settings

Different errors can result from trying to run the same applet in different environments; for example, in IBM VisualAge for Java, in appletviewer (supplied with most Development Kits for Java) or in a Web browser such as Internet Explorer. The differences are to do with different security settings in each environment. You can change the behavior of the environments to allow an applet access to the classes it needs that are stored in package files.

In the following instructions, examples assume use of the Windows platforms. On other platforms, the instructions need slight modification.

For IBM VisualAge for Java:

Change the java.policy file found in
`<vaj_install_dir>\ide\program\lib\security`, where `<vaj_install_dir>` is the directory in which you installed IBM VisualAge for Java.

Refer to "Running WebSphere MQ Java applications under the Java 2 Security Manager" on page 13 for general instructions about changes to this file. For applets, also check for the following changes to the permissions:

1. Comment out the line
   
   ```
   permission java.net.SocketPermission "localhost:1024-", "listen";
   ```

   and replace it with the following line:
   
   ```
   permission java.net.SocketPermission "+", "accept, connect, listen, resolve";
   ```

2. Add the following lines:

   ```
   permission java.util.PropertyPermission "MQJMS_LOG_DIR", "read"
   permission java.util.PropertyPermission "MQJMS_TRACE_DIR", "read"
   permission java.util.PropertyPermission "MQJMS_TRACE_LEVEL", "read"
   permission java.util.PropertyPermission "MQ_JAVA_INSTALL_PATH", "read"
   permission java.util.PropertyPermission "file.separator", "read"
   permission java.util.PropertyPermission "user.name", "read"
   permission java.util.PropertyPermission "com.ibm.mq.jms.cleanup", "read"
   permission java.lang.RuntimePermission "loadLibrary.*"
   ```

Notes:

1. You might need to restart VisualAge for Java if you get the error message Unknown Java Error after repeated tests.

© Copyright IBM Corp. 1997, 2004
2. Make sure that `<install_dir>\java\lib` is in the workspace classpath.

**For appletviewer:**

Find the policy file for your JDK and make the same changes as for IBM VisualAge for Java. For example, in the IBM Developer Kit for Windows, Java Technology Edition, Version 1.3, the `java.policy` file is found in the directory `<jdk_install_dir>\jre\lib\security`, where `<jdk_install_dir>` is the directory where the Developer Kit was installed.

**For a Web browser:**

To achieve consistent behavior for applets within different Web browsers, use the Sun Java plug-in.

1. Install the Sun Java plug-in 1.3.01 or later.
   - From this level, Netscape 6 is also supported.
2. Make the same changes to the `java.policy` file as listed above.
   - The policy file is found in `<java_plugin_install_dir>\lib\security`.
3. Make sure that your HTML applet tags are changed to run with the plug-in.
   - Download and run the Sun HTML Converter v1.3 to make the necessary changes.

---

**Copying package class files**

When a Java program is executed in the context of an applet (which is what is done when appletviewer is executed or a Web browser is used), by default the Java program has significant security restrictions applied to it. One of these restrictions is that all environment variables in effect when the applet is launched are ignored. This includes CLASSPATH.

As a result, unless you make the changes described in "Changing browser security settings" on page 481, when an applet is executed, each and every class that it needs must also be available for download from the same location as the applet code itself.

To achieve this on a Windows system, perform the following steps (non-Windows users need to perform similar tasks):

1. Download and install WINZIP (http://www.winzip.com) or equivalent file unzipping utility
2. Find the files containing the WebSphere MQ Java, or other package, classes that your applet needs.
   - For example, WebSphere MQ base Java classes are in a file called `com.ibm.mq.jar` usually found in the `C:\Program Files\IBM\WebSphere MQ\Java\lib` folder.
3. Using the unzipping utility you installed in step 1 extract all the files in the .jar file into the folder that contains your applet.
   - For the samples supplied with WebSphere MQ Java, the folder to use is `C:\Program Files\IBM\WebSphere MQ\Tools\Java\base`
   - This creates a sub-folder structure `com\ibm`.
4. Run your applet.
Appendix G. Information for SupportPac MA1G

This appendix contains information that is relevant to users of SupportPac MA1G “WebSphere MQ for MVS/ESA™ – WebSphere MQ classes for Java”. MA1G provides support for WebSphere MQ classes for Java from versions of OS/390 not supported by WebSphere MQ Java. It also provides support for CICS and High Performance Java (HPJ).

Users intending to use the WebSphere MQ base Java with CICS Transaction Server for OS/390 must be familiar with:

- Customer Information Control System (CICS) concepts
- Using the CICS Java Application Programming Interface (API)
- Running Java programs from within CICS


Environments supported by SupportPac MA1G

SupportPac MA1G provides support for WebSphere MQ base Java from the following environments:

- OS/390 V2R6 or higher
- Java for OS/390, V1.1.8 or higher
- IBM MQSeries for MVS/ESA, Version 1.2 or higher
- High Performance Java (HPJ)

SupportPac MA1G also provides support for CICS TS1.3 or higher. Support for HPJ in this environment requires OS/390 V2R9 or higher.

SupportPac MA1G does not provide support for JMS.

Obtaining and installing SupportPac MA1G

Obtain SupportPac MA1G from the WebSphere MQ web site http://www.ibm.com/software/integration/mqfamily/. Follow links to Download and then SupportPacs to find the WebSphere MQ Java code.

The following procedure installs the WebSphere MQ classes for Java. The directory used for the installation needs at least 2MB of free storage. In the following, replace /u/joe/mqm with the path name of the directory you choose:

1. Remove any previous installation of this product using the following commands in the OpenEdition shell:
   
   ```
   cd /u/joe
   chmod -fR 700 mqm
   rm -rf mqm
   mkdir mqm
   ```

2. Using FTP binary mode, upload the file ma1g.tar.Z from your workstation to the HFS directory /u/joe/mqm.
Obtaining and installing

3. While in the OpenEdition shell, change to the installation directory `/u/joe/mqm`.
4. Uncompress and untar the file with the command
   ```bash
tar -xpozf ma1g.tar.Z
   ```
5. Set up your CLASSPATH and LIBPATH as described in “Environment variables” on page 10.

Verifying installation using the sample program

To verify installation of MA1G from UNIX System Services (USS), follow the instructions in “Verifying with the sample application” on page 16.

To verify installation of MA1G from CICS Transaction Server:
1. Define the sample application program (MQIVP) to CICS.
2. Define a transaction to run the sample application.
3. Put the queue manager name into the file used for standard input.
4. Run the transaction.

The program output is placed in the files used for standard and error output.

Refer to CICS documentation for more information on running Java programs and setting the input and output files.

Features not provided by SupportPac MA1G

SupportPac MA1G provides a subset of features available to other WebSphere MQ base Java applications. In particular, it does not support the ConnectionPooling feature described in Chapter 7, “Writing WebSphere MQ base Java programs,” on page 67. The following classes and methods are not supported:

- Classes and interfaces
  - MQPoolServices
  - MQPoolServicesEvent
  - MQPoolToken
  - MQSimpleConnectionManager
  - MQPoolServicesEventListener
  - MQConnectionManager
  - ManagedConnection
  - ManagedConnectionFactory
  - ManagedConnectionMetaData

- Methods
  - MQEnvironment.getDefaultConnectionManager()
  - MQEnvironment.setDefaultConnectionManager()
  - MQEnvironment.addConnectionPoolToken()
  - MQEnvironment.removeConnectionPoolToken()
  - The six MQQueueManager constructors which allow a ConnectionManager or MQConnectionManager to be specified.

Attempting to use these classes, interfaces, or methods results in compile-time errors or runtime exceptions.
Running WebSphere MQ base Java applications under CICS

Transaction Server for OS/390

To run a Java application as a transaction under CICS, you must:

1. Define the application and transaction to CICS by using the supplied CEDA transaction.
2. Ensure that the WebSphere MQ CICS adapter is installed in your CICS system. (See WebSphere MQ for z/OS System Setup Guide for details.)
3. Ensure that the JVM environment specified in the DHFJVM parameter of your CICS startup JCL (Job Control Language) includes appropriate CLASSPATH and LIBPATH entries.
4. Initiate the transaction by using any of your normal processes.

For more information on running CICS Java transactions, refer to your CICS system documentation.

Restrictions under CICS Transaction Server

In the CICS Transaction Server for OS/390 environment, only the main (first) thread is allowed to issue CICS or WebSphere MQ calls. It is therefore not possible to share MQQueueManager or MQQueue objects between threads in this environment, or to create a new MQQueueManager on a child thread.

Chapter 8, “Environment-dependent behavior,” on page 95 identifies some restrictions and variations that apply to the WebSphere MQ classes for Java when running against a z/OS or OS/390 queue manager. Additionally, when running under CICS, the transaction control methods on MQQueueManager are not supported. Instead of issuing MQQueueManager.commit() or MQQueueManager.backout(), applications use the JCICS task synchronization methods, Task.commit() and Task.rollback(). The Task class is supplied by JCICS in the com.ibm.cics.server package.
Appendix H. SSL CipherSuites supported by WebSphere MQ

The following table lists the CipherSpecs supported by WebSphere MQ, and their associated CipherSuite names. Specify the CipherSpec name in the SSLCIPH property of the SVRCONN channel on the queue manager. Specify the CipherSuite name:

- In MQEnvironment.sslCipherSuite or MQC.SSL_CIPHER_SUITE_PROPERTY of WebSphere MQ base Java
- Using the setSSLCipherSuite() method of MQConnectionFactory in JMS
- Using the SSLCIPHERSUITE (SCPHS) property from JMSAdmin

The set of supported CipherSuites varies between JSSE providers; those CipherSuites not supported by the IBM implementation of JSSE are marked with an asterisk.

Table 50. CipherSpecs and matching CipherSuites

<table>
<thead>
<tr>
<th>CipherSpec</th>
<th>CipherSuite</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES_SHA_EXPORT</td>
<td>SSL_RSA_WITH_DES_CBC_SHA</td>
</tr>
<tr>
<td>DES_SHA_EXPORT1024</td>
<td>SSL_RSA_EXPORT1024_WITH_DES_CBC_SHA</td>
</tr>
<tr>
<td>NULL_MD5</td>
<td>SSL_RSA_WITH_NULL_MD5</td>
</tr>
<tr>
<td>NULL_SHA</td>
<td>SSL_RSA_WITH_NULL_SHA</td>
</tr>
<tr>
<td>RC2_MD5_EXPORT</td>
<td>SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5</td>
</tr>
<tr>
<td>RC4_56_SHA_EXPORT1024</td>
<td>SSL_RSA_EXPORT1024_WITH_RC4_56_SHA *</td>
</tr>
<tr>
<td>RC4_MD5_US</td>
<td>SSL_RSA_WITH_RC4_128_MD5</td>
</tr>
<tr>
<td>RC4_MD5_EXPORT</td>
<td>SSL_RSA_EXPORT_WITH_RC4_40_MD5</td>
</tr>
<tr>
<td>RC4_SHA_US</td>
<td>SSL_RSA_WITH_RC4_128_SHA</td>
</tr>
<tr>
<td>TRIPLE_DES_SHA_US</td>
<td>SSL_RSA_WITH_3DES_EDE_CBC_SHA</td>
</tr>
</tbody>
</table>
SSL CipherSuites
Appendix I. JMS exception messages

This section lists most common exceptions that can be generated by WebSphere MQ JMS. It does not include all messages that can be written to a trace file. If you receive an exception message not in this list (except in a trace file), or if the cause seems to be an error in WebSphere MQ JMS, contact your IBM service representative.

A JMSException might have an embedded exception that contains a WebSphere MQ reason code. For an explanation of each WebSphere MQ reason code, see the WebSphere MQ Application Programming Reference.

Reading variables in a message
Some messages display text or numbers that vary according to the circumstances giving rise to the message; these are known as message variables. Message variables are indicated in this book by the use of numbers in braces; for example, {0}, [1], and so on.

MQJMS0000 Method [0] has been invoked at an illegal or inappropriate time or if the provider is not in an appropriate state for the requested operation.
Explanation: The normal reason for this exception is that the SSL certificate stores have not been defined. [0] identifies the method that has caused the problem.
User Response: For more information, see "Using Secure Sockets Layer (SSL)" on page 210. For information specific to JMS 1.1, see "Using Secure Sockets Layer (SSL)" on page 253.

MQJMS0002 JMS Client attempted to set invalid clientId on a connection.
Explanation: An application attempted to set the clientId property of a valid connection to null, or attempted to set the clientId property of an invalid connection.
User Response: The clientId property on a connection can only be set once, only to a non-null value, and only before the connection is used. Ensure that the connection is valid and that the clientId value is not null.

MQJMS0003 Destination not understood or no longer valid.
Explanation: The queue or topic may have become unavailable, the application may be using an incorrect connection for the queue or topic, or the supplied destination is not of the correct type for this method.
User Response: Check that WebSphere MQ is still running and the queue manager is available. Check that the right connection is being used for your queue or topic.

MQJMS0004 JMS Client has given JMS Provider a message selector with invalid syntax.
Explanation: The message selector string is empty or contains an invalid value or syntax.
User Response: Check the linked WebSphere MQ exception reason and completion codes for more information.

MQJMS0005 Unexpected end of stream has been reached when a StreamMessage or BytesMessage is being read.
Explanation: The byte stream being read is shorter than the buffer supplied. This can also be caused by receiving a corrupt StreamMessage or BytesMessage.
User Response: Check the length of buffer supplied. Check system event logs for more information.

MQJMS0006 JMS Client attempts to use a data type not supported by a message or attempts to read data in the wrong type.
Explanation: Wrong data types used to read message property types.
User Response: Check that the message received and the properties to be read are of the type expected.

MQJMS0009 JMS Provider is unable to allocate the resources required for a method.
Explanation: Machine resources may be overloaded, the linked exception may give further information.
User Response: Check system resources and load.
### JMS exception messages

<table>
<thead>
<tr>
<th>Code</th>
<th>Exception Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQJMS0010</td>
<td>Operation invalid because a transaction is in progress.</td>
</tr>
<tr>
<td><strong>Explanation:</strong></td>
<td>Wait for the current transaction to complete. See the linked WebSphere MQ exception for further information.</td>
</tr>
<tr>
<td><strong>User Response:</strong></td>
<td>Wait for the current transaction to complete. See the linked WebSphere MQ exception for further information.</td>
</tr>
</tbody>
</table>

| MQJMS0011     | Call to Session.commit resulted in a rollback of the current transaction.               |
| **Explanation:** | The transaction failed resulting in a call to rollback to a safe state. See the linked exception for more information. |

| MQJMS1000     | Failed to create JMS message.                                                         |
| **Explanation:** | Invalid message type or properties were specified when creating a base message.       |
| **User Response:** | Check the linked WebSphere MQ exception Reason and Completion code for more information. |

| MQJMS1001     | Unknown acknowledgement mode [0].                                                      |
| **Explanation:** | Invalid or no parameter [0] set for acknowledgement mode on the session.              |
| **User Response:** | See “Session” on page 393 for the possible values for acknowledgement mode.       |

| MQJMS1004     | Connection closed.                                                                    |
| **Explanation:** | An operation such as start() or stop() has been called on a connection that is already closed. |
| **User Response:** | Ensure that the connection is open before performing any operation.                |

| MQJMS1005     | Unhandled state transition from [0] to [1].                                           |
| **Explanation:** | The state transition is not valid, see log for more information.                    |
| **User Response:** | Check the linked WebSphere MQ exception reason and completion code.             |

| MQJMS1006     | Invalid value for [0]: [1].                                                            |
| **Explanation:** | Invalid value [1] for property [0].                                                   |
| **User Response:** | Check the linked WebSphere MQ exception reason and completion code. See Table 11 on page 49 for a list of valid values for this property. |

| MQJMS1008     | Unknown value of transportType: [0].                                                   |
| **Explanation:** | The value given for transportType could not be used. [0] shows the invalid value.    |
| **User Response:** | See Table 11 on page 49 for a list of valid values for this property.                |

| MQJMS1010     | Not implemented.                                                                       |
| **Explanation:** | The function requested is not implemented. This can be thrown by message acknowledgement, if the session or acknowledgement parameters are invalid or incorrect. |

| MQJMS1011     | Security credentials cannot be specified when using MQ bindings.                       |
| **Explanation:** | The RRS queue does not support a client connection, and bindings connections do not support the specification of security credentials. |
| **User Response:** | Ensure that you do not try to specify security credentials when using a bindings connection. |

| MQJMS1012     | No message listener.                                                                   |
| **Explanation:** | The message listener has stopped or was never started.                                |
| **User Response:** | Restart the message listener and retry.                                               |

| MQJMS1013     | Operation invalid while session is using asynchronous delivery.                        |
| **Explanation:** | You cannot perform the requested operation while the session is actively using asynchronous delivery mode. |
| **User Response:** | For more information, see “Asynchronous delivery” on page 208. For information specific to JMS 1.1, see “Asynchronous delivery” on page 248. |

| MQJMS1014     | Operation invalid for identified producer.                                             |
| **Explanation:** | The QueueSender.send method has been performed on an identified QueueSender, which contradicts the JMS specification |
| **User Response:** | See “QueueSender” on page 387 and the JMS specification (http://java.sun.com/products/jms/docs.html) for further information. |

| MQJMS1015     | Unknown value of targetClient: [0].                                                    |
| **Explanation:** | The value for the targetClient property set by the application for this destination is not recognized by WebSphere MQ JMS. |
| **User Response:** | See “Sending a message” on page 204 for valid values of the targetClient property.   |

| MQJMS1017     | Non-local MQ queue not valid for receiving or browsing.                               |
| **Explanation:** | An attempt was made to perform an inappropriate operation on a non-local queue.       |
| **User Response:** | Check the queue properties.                                                          |

| MQJMS1018     | No valid connection available.                                                         |
| **Explanation:** | The queue is busy, there are network problems or a connection has not been defined for the object. |
| **User Response:** | Create a valid connection for this operation.                                         |
MQJMS1019 Invalid operation for non-transacted session.
Explanation: Commit is not allowed on a session that is not transacted.
User Response: Check the linked IllegalStateException for more information. See "Session" on page 393 for further information.

MQJMS1020 Invalid operation for transacted session.
Explanation: Invalid acknowledgement mode for a transacted session. Acknowledge and Recover are not valid operations in transacted sessions.
User Response: See "Session" on page 393 for further information.

MQJMS1021 Recover failed: unacknowledged messages might not get redelivered.
Explanation: The system was unable to recover from a failure.
User Response: Consult the linked exception to determine why the call to recover failed.

MQJMS1022 Failed to redirect message.
Explanation: When performing asynchronous delivery, WebSphere MQ JMS attempted to redirect the message to the backout queue. No backout queue was defined.
User Response: Ensure that the backout queue is defined. Also, investigate why WebSphere MQ JMS was attempting to redirect the message. It might do so in response to a failing MessageListener implementation.

MQJMS1023 Rollback failed.
Explanation: The system was unable to rollback to a safe state.
User Response: Check the linked WebSphere MQ Exception reason and completion codes for further information.

MQJMS1024 Session closed.
Explanation: The session timed out or was closed; or either the connection or the queue manager was closed, implicitly closing the session.
User Response: Restart the session, and check all required resources are available.

MQJMS1025 Failed to browse message.
Explanation: No message was available for browsing. There may be no message on the Queue.
User Response: Check the linked WebSphere MQ Exception reason and completion codes. Check that a message is available for browsing.

MQJMS1026 ExceptionListener threw exception: [0].
User Response: Check linked exceptions for further information.

MQJMS1027 Failed to reconstitute destination from {0}.
Explanation: A message has been received which contains invalid destination information in the RFH2 header.
User Response: Ensure that any messages being sent by non-JMS applications have correctly formatted destination information. In the case of RFH2 headers, pay special attention to the “Rto” (reply to) and “Dst” (destination) elements of the XML portion of the header. Valid destination strings must start either “queue” or “topic”.

MQJMS1028 Element name is null.
Explanation: A null name string was passed to one of the “get value by name” methods of MapMessage.
User Response: Ensure that all name strings being used to retrieve values are non-null.

MQJMS1029 Property name is null.
Explanation: The itemExists method of MapMessage was invoked with a null item name; or a null name string was used as an argument to a method which retrieves property values by name from a JMS message.
User Response: Ensure that the name strings indicated do not have null values.

MQJMS1031 An internal error has occurred. Please contact your system administrator.
Explanation: Internal Error.
User Response: Contact your IBM representative.

MQJMS1032 close() failed because of [0]
Explanation: Internal Error. [0] indicates the reason for the error.
User Response: Contact your IBM representative.

MQJMS1033 start() failed because of [0].
Explanation: [0] indicates why the session failed to start.
User Response: Contact your IBM representative.

MQJMS1034 MessageListener threw: [0].
Explanation: When performing asynchronous delivery, the onMessage() method of the application’s MessageListener failed with a Throwable. WebSphere MQ JMS tries to redeliver or requeue the message.
User Response: Do not throw Throwables from the onMessage() method of a MessageListener.
JMS exception messages

MQJMS1035 Cannot transmit non-MQ JMS messages.
Explanation: Wrong message type used. This is a possible internal problem.
User Response: Check the message type. Contact your IBM representative if there appears to be an internal error.

MQJMS1036 Failed to locate resource bundle.
Explanation: The resource bundle is either not present or not in the application’s classpath.
User Response: Check that the classpath includes the location of property files.

MQJMS1038 Failed to log error.
Explanation: Log settings may be incorrect, see the linked LogException.
User Response: Check log settings are correct.

MQJMS1039 Trace file does not exist
Explanation: Trace settings may be incorrect.
User Response: Check trace settings and trace file existence. See "Tracing programs" on page 38 for more information on Trace.

MQJMS1040 Failed to connect to Trace stream.
Explanation: Trace settings may be incorrect.
User Response: See "Tracing programs" on page 38 for more information on Trace.

MQJMS1041 Failed to find system property {0}.
Explanation: The system property specified in {0} does not exist or was not found in the application’s classpath.
User Response: Check the classpath settings and the product installation.

MQJMS1042 Invalid delivery mode.
Explanation: Either an invalid value was specified for the delivery mode of a message producer, or an invalid delivery mode value was specified when publishing a message.
User Response: Check to ensure that the value specified is a valid enumeration for delivery mode.

MQJMS1043 JNDI failed due to {0}.
Explanation: {0} gives further information.
User Response: Check settings for LDAP, JNDI, and in the JMSAdmin.config file.

MQJMS1044 String is not a valid hexadecimal number - {0}.
Explanation: An attempt was made to specify a group ID or correlation ID which starts with the prefix "ID:" but is not followed by a well-formed hex value; or an attempt was made to receive a message which contains an RFH2 property of type bin.hex that does not have a well-formed hex value.
User Response: Ensure that a valid hex value always follows the "ID:" prefix when setting group ID or correlation ID values. Ensure that any RFH2 headers generated by non-JMS applications are well-formed.

MQJMS1045 Number outside of range for double precision S/390 Float {0}.
Explanation: This is a z/OS and OS/390 specific error.

MQJMS1046 The character set {0} is not supported.
Explanation: An attempt was made to send or receive a map message, stream message or text message whose body is encoded using a character set not supported by the JVM. In the case of text messages, this exception may be thrown when the body of the message is first queried, rather than at receive time.
User Response: Only set character encoding on a message to values known to be available to the receiving application.

MQJMS1047 The map message has an incorrect format.
Explanation: A map message was received, but its RFH2 header information is badly formatted.
User Response: Ensure any non-JMS applications are building well-formed RFH2 header information for inclusion in map messages.

MQJMS1048 The stream message has an incorrect format.
Explanation: A stream message was received, but its RFH2 header information is badly formatted.
User Response: Ensure any non-JMS applications are building well-formed RFH2 header information for inclusion in stream messages.

MQJMS1049 The JMS client attempted to convert a byte array to a String.
Explanation: Attempting to receive a byte array from a stream message using the readString method.
User Response: Either use the appropriate method to receive the data, or format the data placed into the stream message correctly.

MQJMS1050 The MQRFH2 header has an incorrect format.
Explanation: Receiving a message with a badly formed RFH2 header.
User Response: Ensure that any non-JMS applications building messages with RFH2 headers create well-formed RFH2 headers.
MQJMS1053 Invalid UTF-16 surrogate detected [0].
Explanation: An invalid UTF-16 surrogate character has been encountered as part of a topic name or RFH2 property.
User Response: Ensure that, when specifying UTF-16, topic names or RFH2 properties are well-formed.

MQJMS1054 Invalid XML escape sequence detected [0].
Explanation: An invalid XML escape sequence has been encountered in the RFH2 header of a received message.
User Response: Ensure that only valid XML escape sequences are placed into any RFH2 headers built by non-JMS applications.

MQJMS1055 The property or element in the message has incompatible datatype [0].
Explanation: Attempting to retrieve a property from a JMS message using a accessor method which specifies an incompatible type. For example, attempting to retrieve an integer property using the getBooleanProperty method.
User Response: Use an accessor method defined by the JMS specification as being able to retrieve property values of the required type.

MQJMS1056 Unsupported property or element datatype [0].
Explanation: This error is caused by one of the following:
1. Attempting to set a property of a JMS message using an object which is not one of the supported types.
2. Attempting to set or receive a message whose RFH2 contains a element representing a property which does not have a valid type associated with it.
User Response: Ensure that when setting message properties, an object type described as being valid in the JMS specification is used. If this exception occurs when receiving a message containing an RFH2 header sent by a non-JMS application, ensure that the RFH2 header is well-formed.

MQJMS1057 Message has no session associated with it.
Explanation: An attempt was made to acknowledge a message on a session which is not in an open state.
User Response: Ensure that the session associated with the message has been correctly opened. Check that the session has not been closed.

MQJMS1058 Invalid message property name [0].
Explanation: Attempting to set a property that either does not have a valid property name, or is not a settable property.
User Response: Ensure that the property name used is a valid property name in accordance with the JMS specification. If the property name refers to a JMS or provider-specific extension property, ensure that this property is settable.

MQJMS1059 Fatal error - UTF8 not supported.
Explanation: The Java runtime environment you are using does not support the UTF-8 character encoding. JMS requires support for this encoding to perform some operations.
User Response: Consult the documentation and or provider of your Java runtime environment to determine how to obtain support for the UTF-8 character encoding.

MQJMS1060 Unable to serialize object.
Explanation: An attempt has been made to serialize an ObjectMessage which contains a non-serializable object.
User Response: Ensure that ObjectMessages only contain serializable objects. If the object placed inside an ObjectMessage references other objects, these must also be serializable.

MQJMS1061 Unable to deserialize object.
Explanation: De-serialization of an ObjectMessage failed.
User Response: Ensure that the ObjectMessage being received contains valid data. Ensure that the class files representing object data contained within the ObjectMessage are present on the machine deserializing the ObjectMessage. If the object contained within the ObjectMessage references other objects, ensure that these class files are also present.

MQJMS1066 Invalid message element name [0].
Explanation: Attempting to set a message property using either an invalid property name, or the name of a property which cannot have its value set.
User Response: Ensure that the property name specified conforms to the JMS specification. If the property name supplied is that of a JMS property, or a vendor specific extension, ensure that this property name is settable.

MQJMS1067 Timeout invalid for MQ.
Explanation: An attempt was made to invoke the receive method on either a QueueReceiver or TopicSubscriber method, specifying a long timeout value which is not valid.
User Response: Ensure the timeout value specified is not negative and not greater than the value of Integer.MAX_VALUE.

Appendix I. JMS exception messages 493
MQJMS1068 Failed to obtain XAResource.
Explanation: JMS failed to create an XA Queue resource due to an error.
User Response: See the linked XAException for more information.

MQJMS1072 Could not inquire upon queue manager name.
Explanation: In createConnectionConsumer() or createDurableConnectionConsumer(), JMS could not determine the name of the queue manager.
User Response: Check your queue manager error logs for problems which may cause this. If there are no other error conditions, contact your IBM representative.

MQJMS1073 Specified MQ Queue is neither a QLOCAL nor a QALIAS.
Explanation: createConnectionConsumer() was called, but a queue of the wrong type was specified. Only QALIAS and QLOCALs can be used with the ConnectionConsumer feature.
User Response: Specify a queue of the correct type.

MQJMS1074 Unable to process null message.
Explanation: Internal error in WebSphere MQ JMS.
User Response: Contact your IBM representative.

MQJMS1075 Error writing dead letter header.
Explanation: JMS attempted to requeue a message to the dead letter queue, but could not construct a dead letter header.
User Response: Use the linked exception to determine the cause of this error.

MQJMS1076 Error reading dead letter header.
Explanation: JMS attempted to interpret a message with a dead letter header, but encountered a problem.
User Response: Use the linked exception to determine the cause of this error.

MQJMS1077 Connection and Destination mismatch.
Explanation: An operation was requested, but the Destination class is incompatible with the Connection class. Topics cannot be used with QueueConnections and Queues cannot be used with TopicConnections.
User Response: Supply a suitable Destination. This may represent an internal error condition in JMS; in this case contact your IBM representative.

MQJMS1078 Invalid Session object.
Explanation: The JMS ConnectionConsumer feature attempted to deliver a batch of messages to a Session. However, the Session contained in the ServerSessionPool object returned by the ServerSessionPool was not a WebSphere MQ JMS Session.
User Response: This is an error in the ServerSessionPool. If you have supplied a ServerSessionPool, check its behavior. In a J2EE application server, this may represent an error in the application server; in which case, refer to your application server’s documentation.

MQJMS1079 Unable to write message to dead letter queue.
Explanation: JMS attempted to requeue a message to the dead letter queue, but failed.
User Response: Use the linked exception to determine the cause of this error. If there is no linked exception, check that the queue manager has a defined dead letter queue. Once JMS has sent a message to the dead letter queue, the reason code stored in the message’s DLH can be used to determine why the message was dead-lettered.

MQJMS1080 No Backout-Requeue queue defined.
Explanation: JMS encountered a message which has been backed out more than the queue’s Backout Threshold, however the queue doesn’t have a Backout-Requeue queue defined.
User Response: Define a Backout-Requeue queue for the queue, or set the Backout Threshold to zero to disable poison message handling. Investigate the repeated backouts.

MQJMS1081 Message requeue failed.
Explanation: JMS found an error when requeuing a message which has been backed out more than the queue’s Backout Threshold.
User Response: Use the linked exception to determine the cause of this error. Investigate the repeated backouts.

MQJMS1082 Failure while discarding message.
Explanation: JMS encountered an error while discarding a message, or while generating an exception report for a message to be discarded.
User Response: Use the linked exception to determine the cause of this error.

MQJMS1083 Invalid message batch size (must be >0).
Explanation: An invalid batch size parameter was passed to createConnectionConsumer() or createDurableConnectionConsumer().
User Response: Set a batch size greater than zero. In a J2EE application server, this may represent an error in the application server. Refer to your application server’s documentation.

MQJMS1084 Null ServerSessionPool has been provided.
Explanation: The ServerSessionPool specified on createConnectionConsumer() or createDurableConnectionConsumer() was null.
User Response: Set an appropriate ServerSessionPool. In a J2EE application server, this may represent an error
in the application server. Refer to your application server’s documentation.

**MQJMS1085 Error writing RFH.**
Explanation: JMS attempted to construct an RFH message header, but encountered an error.
User Response: Use the linked exception to determine the cause of this error.

**MQJMS1086 Error reading RFH.**
Explanation: JMS encountered an error while parsing an RFH message header.
User Response: Use the linked exception to determine the cause of this error.

**MQJMS1087 Unrecognized or invalid RFH content.**
Explanation: JMS expected to find an RFH message header, but found it to be missing, malformed or lacking required data.
User Response: Investigate the source of the message. This may represent an internal error condition in JMS; in this case, contact your IBM representative.

**MQJMS1088 Mixed-domain consumers acting on the same input is forbidden.**
Explanation: A point-to-point ConnectionConsumer is using the subscriber queue of a publish/subscribe ConnectionConsumer.
User Response: Do not attempt to access subscriber queues using the point-to-point ConnectionConsumer facilities of JMS. Check your TopicConnectionFactory and Topic objects to make sure they are not using a QLOCAL intended for use by point-to-point applications as a subscriber queue.

**MQJMS1089 Exception occurred reading message body: [0].**
Explanation: JMS encountered an exception while reading data from a message. The message being read is likely to be a response message from the publish/subscribe broker.
User Response: Use the linked exception to determine the cause of this error.

**MQJMS1111 JMS 1.1 The required queues or publish/subscribe services are not set up: [0].**
Explanation: The required WebSphere MQ setup for the messaging domain is not complete.
User Response: For the point-to-point messaging, make sure that you have started the queue manager and, if your JMS application is connecting as a client application, make sure that you have started a listener for the correct port. For publish/subscribe messaging, make sure that you have done the post installation setup, as described in “Additional setup for publish/subscribe mode” on page 26.

**MQJMS1112 JMS 1.1 Invalid operation for a domain specific object.**
Explanation: A JMS application attempted to perform an operation on domain specific object, but the operation is valid only for the other messaging domain.
User Response: Make sure that the JMS objects used by your application are relevant for the required messaging domain. If your application uses both messaging domains, consider using domain independent objects throughout the application.

**MQJMS1113 JMS 1.1 Invalid attribute for a domain specific object.**
Explanation: A JMS application attempted to set an attribute of a domain specific object, but the attribute is valid only for the other messaging domain.
User Response: Make sure that the JMS object types used by your application are relevant for the required messaging domain. If your application uses both messaging domains, consider using domain independent objects throughout the application.

**MQJMS2000 Failed to close MQ queue.**
Explanation: JMS attempted to close a WebSphere MQ queue, but encountered an error. The queue may already be closed, or another thread may be performing an MQGET while close() is called.
User Response: Use the linked exception to determine the cause of this error. You may be able to perform the close() later.

**MQJMS2001 MQQueue reference is null.**
Explanation: JMS attempted to perform some operation on a null MQQueue object.
User Response: Check your system setup, and that all required queue names have been specified. This may represent an internal error condition in JMS; in this case, contact your IBM representative.

**MQJMS2002 Failed to get message from MQ queue.**
Explanation: JMS attempted to perform an MQGET; however WebSphere MQ reported an error.
User Response: Use the linked exception to determine the cause of this error.

**MQJMS2003 Failed to disconnect queue manager.**
Explanation: JMS encountered an error while attempting to disconnect.
User Response: Use the linked exception to determine the cause of this error.

**MQJMS2004 MQQueueManager reference is null.**
Explanation: JMS attempted to perform an operation on a null MQQueueManager object.
User Response: Check that the relevant object has not been closed. This may represent an internal error.
JMS exception messages

condition in JMS; in this case, contact your IBM representative.

\textbf{MQJMS2005 Failed to create MQQueueManager for }\{0\}.  
\textbf{Explanation:} JMS could not connect to a queue manager. \{0\} gives the name of the queue manager.  
\textbf{User Response:} Use the linked exception to determine the cause of this error. Check the queue manager is running and, if using client attach, that the listener is running and the channel, port and hostname are set correctly. If no queue manager name has been specified, check that the default queue manager has been defined.

\textbf{MQJMS2006 MQ problem: }\{0\}.  
\textbf{Explanation:} JMS encountered some problem with WebSphere MQ. \{0\} describes the problem.  
\textbf{User Response:} Use the included text and linked exception to determine the cause of this error.

\textbf{MQJMS2007 Failed to send message to MQ queue.}  
\textbf{Explanation:} JMS attempted to perform an MQPUT; however WebSphere MQ reported an error.  
\textbf{User Response:} Use the linked exception to determine the cause of this error.

\textbf{MQJMS2008 Failed to open MQ queue.}  
\textbf{Explanation:} JMS attempted to perform an MQOPEN; however WebSphere MQ reported an error.  
\textbf{User Response:} Use the linked exception to determine the cause of this error. Check that the specified queue and queue manager are defined correctly.

\textbf{MQJMS2009 MQQueueManager.commit() failed.}  
\textbf{Explanation:} JMS attempted to perform an MQCMIT; however WebSphere MQ reported an error.  
\textbf{User Response:} Use the linked exception to determine the cause of this error.

\textbf{MQJMS2010 Unknown value for MQ queue definitionType: }\{0\}.  
\textbf{Explanation:} Unable to delete the temporary queue as the definitionType is not valid.  
\textbf{User Response:} Check the setting of definitionType.

\textbf{MQJMS2011 Failed to inquire MQ queue depth.}  
\textbf{Explanation:} WebSphere MQ JMS is unable to tell how many messages are on the queue.  
\textbf{User Response:} Check that the queue and queue manager are available.

\textbf{MQJMS2012 XACLOSE failed.}  
\textbf{Explanation:} See linked XAException for more details.

\textbf{MQJMS2013 Invalid security authentication supplied for MQQueueManager.}  
\textbf{Explanation:} Bad username or password or both. In bindings mode, a supplied user ID does not match the logged in user ID.  
\textbf{User Response:} Check that the user IDs used by WebSphere MQ are all assigned to the relevant groups and given appropriate user permissions.

\textbf{MQJMS3000 Failed to create a temporary queue from }\{0\}.  
\textbf{Explanation:} Creation of temporary queue failed.  
\textbf{User Response:} See linked exception for more information. Check that the TemporaryModel parameter against the QueueConnectionFactory is set to a valid model queue.

\textbf{MQJMS3001 Temporary queue already closed or deleted.}  
\textbf{Explanation:} Temporary queue no longer exists or is equal to null.  
\textbf{User Response:} Check to see that the queue has been created, and that the session is still available.

\textbf{MQJMS3002 Temporary queue in use.}  
\textbf{Explanation:} Another program is using the queue.  
\textbf{User Response:} Wait for the temporary queue to become free or create another.

\textbf{MQJMS3003 Cannot delete a static queue.}  
\textbf{Explanation:} Attempted to delete a queue of type static, where a temporary queue was expected.  
\textbf{User Response:} Check the expected queue type for deletion.

\textbf{MQJMS3004 Failed to delete temporary queue.}  
\textbf{Explanation:} The temporary queue may be persistent or busy.  
\textbf{User Response:} See the linked WebSphere MQ exception for more details. Wait if the queue is busy, or delete the queue manually if it is persistent.

\textbf{MQJMS3005 Publish/Subscribe failed due to }\{0\}.  
\textbf{Explanation:} General error: \{0\} shows the reason.  
\textbf{User Response:} Check the linked WebSphere MQ Exception reason and completion codes for more information. It is possible that the broker and queue manager versions are incompatible.

\textbf{MQJMS3006 Topic reference is null.}  
\textbf{Explanation:} Topic supplied to a publisher is null.  
\textbf{User Response:} Use non-null values.
<table>
<thead>
<tr>
<th>Exception Code</th>
<th>Explanation</th>
<th>User Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQJMS3008</td>
<td>Failed to build command [0].</td>
<td>Broker message command parameters incorrect. Check linked exception for cause.</td>
</tr>
<tr>
<td>MQJMS3009</td>
<td>Failed to publish command to MQ queue.</td>
<td>Invalid command, queue unavailable or broker errors. Check linked WebSphere MQ exception reason and completion codes for more information.</td>
</tr>
<tr>
<td>MQJMS3010</td>
<td>Failed to build publish message.</td>
<td>Unable to build the base message for the broker. See the linked WebSphere MQ Exception for further details. Check settings and parameters are all correct. See Chapter 11, “Writing WebSphere MQ JMS publish/subscribe applications,” on page 213 for more information.</td>
</tr>
<tr>
<td>MQJMS3011</td>
<td>Failed to publish message to MQ queue.</td>
<td>See linked Exception for more information. Check settings and parameters are all correct. See Chapter 11, “Writing WebSphere MQ JMS publish/subscribe applications,” on page 213 for more information.</td>
</tr>
<tr>
<td>MQJMS3013</td>
<td>Failed to store admin. entry.</td>
<td>An add to the admin or status queue failed due to duplication or some other error. See any linked exception for more information. Check for duplicates and retry.</td>
</tr>
<tr>
<td>MQJMS3014</td>
<td>Failed to open subscriber queue [0].</td>
<td>See linked exception for more information.</td>
</tr>
<tr>
<td>MQJMS3017</td>
<td>Failed to delete subscriber queue [0].</td>
<td>Gives the queue name. See linked exception for more information. See Chapter 11, “Writing WebSphere MQ JMS publish/subscribe applications,” on page 213 for more information on solving publish/subscribe problems.</td>
</tr>
<tr>
<td>MQJMS3018</td>
<td>Unknown durable subscription [0].</td>
<td>Could not locate the given subscription. For example, during an unsubscribe request. See Chapter 11, “Writing WebSphere MQ JMS publish/subscribe applications,” on page 213 for more information.</td>
</tr>
<tr>
<td>MQJMS3020</td>
<td>TemporaryTopic out of scope.</td>
<td>The current connection ID does not match the connection that created the temporary topic.</td>
</tr>
<tr>
<td>MQJMS3021</td>
<td>Invalid subscriber queue prefix: [0].</td>
<td>The name specified is not valid. It must begin with SYSTEM,JMS,D for durable subscriptions or SYSTEM,JMS,ND for non-durable subscriptions. See Chapter 11, “Writing WebSphere MQ JMS publish/subscribe applications,” on page 213 for naming conventions.</td>
</tr>
<tr>
<td>MQJMS3022</td>
<td>Durable re-subscribe must use same subscriber queue; specified: [0], original: [1].</td>
<td>Unable to get a subscription due to wrong queue manager or queue. Check settings.</td>
</tr>
<tr>
<td>MQJMS3023</td>
<td>Subscription has an active TopicSubscriber.</td>
<td>Can be caused by a queue open problem or if a subscription already exists on the JVM. If running in WebSphere Application Server there can be other causes. See linked exception, if set, for more information. Check settings.</td>
</tr>
<tr>
<td>MQJMS3024</td>
<td>Illegal use of uninitialized clientId.</td>
<td>The clientId in the connection has not been set. Set the clientId before attempting to perform any operation.</td>
</tr>
<tr>
<td>MQJMS3025</td>
<td>TemporaryTopic in use.</td>
<td>Something else is currently using the topic. Wait until the topic is free or create another topic. Ensure subscribers de-register when finished.</td>
</tr>
<tr>
<td>MQJMS3026</td>
<td>QueueSender is closed.</td>
<td>Open or re-open the queue sender if required.</td>
</tr>
<tr>
<td>MQJMS3027</td>
<td>Local transactions not allowed with XA sessions.</td>
<td>A call pertaining to a local transaction was made on a Session involved with XA-coordinated transactions. This typically represents an error in an application server. Consult your application server’s documentation and any error logs.</td>
</tr>
</tbody>
</table>
JMS exception messages

MQJMS3028 TopicPublisher is closed.
User Response: Open or reopen the topic publisher if required.

MQJMS3029 Enlist failed (see linked Exception).
Explanation: JTSXA.enlist threw an exception that was caught by JMS.
User Response: Check the linked WebSphere MQ Exception reason and completion codes for more information. Contact your IBM representative.

MQJMS3031 clientId cannot be set after connection has been used.
Explanation: The clientId of a connection can be set only once and only before the connection is used.
User Response: Set the clientId before using the connection.

MQJMS3032 Resetting the clientId is not allowed.
Explanation: The clientId of a connection can be set only once and only before the connection is used.
User Response: Set the clientId before using the connection.

MQJMS3033 QueueReceiver is closed.
User Response: Open or reopen the receiver.

MQJMS3034 TopicSubscriber is closed.
User Response: Open or reopen the TopicSubscriber.

MQJMS3036 Broker side message selection valid only when using WebSphere MQ Integrator broker.
Explanation: Broker version and message selection are not consistent.
User Response: Ensure the broker version has been set in theConnectionFactory. Use the method setBrokerVersion(JMSC.MQJMS_BROKER_V2) on the ConnectionFactory for WebSphere MQ Integrator or WebSphere MQ Event Broker.

MQJMS3037 Message Producer is closed.
Explanation: Either or both of the session and connection are closed.
User Response: Check to ensure that the session and connection are both available.

MQJMS3038 Message Consumer is closed.
Explanation: Either or both of the session and connection are closed.
User Response: Check to ensure that the session and connection are both available.

MQJMS3039 Illegal use of null name.
Explanation: Durable connection consumers must be named.
User Response: Check for null values.

MQJMS3040 Invalid broker control message content: [ ]
Explanation: [ ] explains further.
User Response: Check the broker documentation for message content information.

MQJMS3042 Unrecognized message from Pub/Sub Broker.
Explanation: The message received from the broker was not of a recognized or supported format.
User Response: Check that the broker you are using is supported and refer to broker documentation for settings.

MQJMS3044 Cleanup level of NONE requested.
Explanation: Cleanup requested while cleanupLevel set to NONE.
User Response: Set cleanupLevel property to an appropriate value.

MQJMS3047 Subscription store type not supported by queue manager.
Explanation: Not an MQSPIQueue manager or deferred message not supported.
User Response: Possible incompatibility between queue manager version and broker. Specify a different type of subscription store or upgrade the queue manager. For more information, see “Subscription stores” on page 227. For information specific to JMS 1.1, see “Subscription stores” on page 246.

MQJMS3048 Incorrect subscription store type.
Explanation: Subscription store changed within TopicConnection.
User Response: Contact your IBM representative.

MQJMS3049 Incorrect subscription type for this subscription store.
Explanation: TopicSubscriber was created with a different SUBSTORE setting than current TopicConnection.
User Response: Ensure TopicSubscribers are only used during the lifetime of their parent TopicConnection. For more information, see “Subscription stores” on page 227. For information specific to JMS 1.1, see “Subscription stores” on page 246.
MQJMS4009 Context is not empty.
Explanation: Error deleting Context due to context not being empty.
User Response: Remove context contents before trying delete.

MQJMS4096 Binding non-administrable or not found.
Explanation: From JMSAdmin, an object was specified on the command line that either does not exist, or is not an object that JMSAdmin can administer.
User Response: Specify a valid object on the JMSAdmin command line.

MQJMS4097 Context not found.
Explanation: Could not find a context to match the name given.
User Response: Ensure the correct context name is specified.

MQJMS4104 Object is inactive, so cannot perform directory operations.
Explanation: The JNDI service is inactive.
User Response: See Chapter 5, “Using the WebSphere MQ JMS administration tool,” on page 41 for JMSAdmin and JNDI information.

MQJMS4106 Object is not a WebSphere MQ JMS administered object.

MQJMS4111 Unable to create context.
Explanation: Administration service failed.
User Response: Check LDAP and JNDI settings.

MQJMS4112 Unable to create a valid object, please check the parameters supplied.
Explanation: Consistency check failed.
User Response: Contact your IBM representative.

MQJMS4113 Unable to bind object.
Explanation: Administration service bind or copy or move operation failed.
User Response: Check that you have correctly set up your JNDI provider.

MQJMS4115 An invalid name was supplied.
Explanation: JMSAdmin error. An invalid name was supplied when trying to delete a context.
User Response: Refer to Chapter 5, “Using the WebSphere MQ JMS administration tool,” on page 41 for more about using JMSAdmin.

MQJMS4121 Cannot open configuration file.
Explanation: Configuration file may not exist.
User Response: Check MQ_JAVA_INSTALL_PATH environment variable exists and points to the installation directory of the base Java classes.

MQJMS4127 Invalid property in this context.
Explanation: JMSAdmin object value is invalid in the current context.
User Response: See Chapter 5, “Using the WebSphere MQ JMS administration tool,” on page 41 for more about JMSAdmin.

MQJMS4130 Context not found or unremovable.
Explanation: The specified child context could not be deleted.
User Response: Ensure the correct context name was specified.

MQJMS4131 Expected and actual object types do not match.
Explanation: Requested and retrieved objects are of different types.
User Response: Check that you have specified the correct object type.

MQJMS4132 Client-bindings attribute clash.
Explanation: Client properties specified for a bindings connection.
User Response: Ensure the ConnectionFactory properties are correct.

MQJMS4133 ExitInit string supplied without Exit string.
Explanation: ExitInit string supplied but Exit is not set.
User Response: Set appropriate exit, or unset ExitInit string.

MQJMS4137 Unable to create a WebSphere MQ specific class. The WebSphere MQ classes may not have been installed or added to the classpath.
User Response: Check WebSphere Application Server installation and classpath variable.

MQJMS4139 Invalid authentication type supplied - using ‘none’.
Explanation: AdminService JNDI initialization parameters contain an invalid authorization scheme, so “none” is used as the value instead.
User Response: See Chapter 5, “Using the WebSphere MQ JMS administration tool,” on page 41 for more information.
## JMS exception messages

**MQJMS5053** ***No broker response. Please ensure that the broker is running.***

**Explanation:** Possible causes:
1. Broker is not running.
2. You are using BrokerVersion=V2 in your TopicConnectionFactory with the MQSeries Publish/Subscribe broker, which does not support this.
3. The Broker has rejected the Publication or Subscription and placed it on the SYSTEM.DEAD.LETTER.QUEUE

**User Response:** Ensure that your broker is running. Check the system event log for broker error messages. Check that the broker supports the BrokerVersion you are using. Check the SYSTEM.DEAD.LETTER.QUEUE for rejected messages.

**MQJMS5054** The broker appears to be running, but the message did not arrive.

**Explanation:** Thrown by Installation Verification Test when the subscriber fails to receive the published message.

**User Response:** Check that you have set up the broker correctly. Check system event logs for broker error messages. Check the SYSTEM.DEAD.LETTER.QUEUE for messages rejected by the broker.

**MQJMS5060** Unable to connect to queue manager.

**Explanation:** Thrown by Installation Verification Test.

**User Response:** Check that the queue manager is running and that its name is specified correctly in the IVTTest parameters.

**MQJMS5061** Unable to access broker control queue on queue manager.

**User Response:** Check that the control queue exists. The default name is SYSTEM.BROKER.CONTROL.QUEUE.

**MQJMS6039** An exception occurred while loading the minimal client security implementation.

**User Response:** Contact your IBM representative.

**MQJMS6040** An unexpected exception in minimal client, exception = {0}.

**Explanation:** An unusual or unexpected exception occurred at the minimal client. [0] gives more details.

**User Response:** Contact your IBM representative.

**MQJMS6041** An exception occurred while attempting to load socket factory class [0], exception: <(1)>.

**Explanation:** Either a ClassNotfoundException, an InstantiationException or an IllegalAccessException occurred while trying to load a particular IMBSocketFactory. [0] gives the name of the class.

**User Response:** Contact your IBM representative.

**MQJMS6042** The broker indicated an error on the minimal client connection.

**User Response:** Refer to JMS or broker documentation. Contact your IBM representative.

**MQJMS6043** Connector.send was called with an illegal message value.

**Explanation:** Connector.send was called with an illegal message value.

**User Response:** See “Using topics” on page 221 for more information.

**MQJMS6044** An illegal value was encountered for a field, value = {0}.

**Explanation:** [0] shows the illegal value.

**User Response:** See Table 38 on page 457 for a list of properties and their possible values.

**MQJMS6045** An unexpected internal error occurred in the minimal client.

**Explanation:** Internal problem.

**User Response:** Contact your IBM representative.

**MQJMS6046** A bytes message operation was requested on something that is not a bytes message.

**Explanation:** The wrong message type was found.

**User Response:** Check message type before performing type specific operations.
| MQJMS6068 | A text message operation was requested on something that is not a text message. |
| MQJMS6069 | A stream message operation was requested on something that is not a stream message. |
| MQJMS6070 | A map message operation was requested on something that is not a map message. |
| MQJMS6071 | The broker sent an invalid message during authentication. |
| MQJMS6072 | The broker requested an unavailable protocol during authentication. |
| MQJMS6073 | Minimal client connection rejected because of authentication failure. |
| MQJMS6074 | No QOP available in the minimal client. |
| MQJMS6075 | An attempt was made to write an invalid object type of class [0]. |
| MQJMS6076 | An exception occurred while attempting to load thread pooling support, exception = [0]. |

MQJMS6081: An attempt was made to read from a Stream message before a previous read has completed.

MQJMS6083: An exception occurred while initializing a thread pool instance, exception = [0].

MQJMS6085: No ExceptionListener has been set.

MQJMS6088: The client-side connection monitor is terminating.

MQJMS6090: Attempted to synchronously receive on a MessageConsumer for which a listener is active.

MQJMS6091: An IOException occurred when starting or stopping delivery on the connection, exception = [0].

MQJMS6093: An exception occurred during synchronous receive, exception = [0].

MQJMS6096: A JMSPriority level of [0] is outside the range specified in JMS.

MQJMS6097: The specified JMSMessageID, [0], is invalid.
### JMS exception messages

“JMS messages,” on page 257 for more information on message IDs.

---

**MQJMS6105** No more client parameter changes allowed.

**Explanation:** An attempt was made to set a SessionConfig parameter when no more changes are allowed. Internal error.

**User Response:** Contact your IBM representative.

---

**MQJMS6106** An exception occurred when initializing parameter {0}, exception {1}.

**Explanation:** {0} identifies the failing parameter and {1} the caught exception.

**User Response:** Contact your IBM representative.

---

**MQJMS6115** An exception occurred while creating the TopicConnection, exception {0}.

**Explanation:** {0} gives details of the exception.

**User Response:** Contact your IBM representative.

---

**MQJMS6116** This operation is not permitted on an entity that is closed.

**Explanation:** An operation was requested on a closed publisher, session, or connection.

**User Response:** Ensure that the publisher, session, or connection is open before trying this operation.

---

**MQJMS6117** The {0} implementation of Topic is not supported.

**Explanation:** The Topic instance passed to a TopicPublisher or TopicSession method has an unsupported run-time implementation. {0} gives the class name of the unsupported implementation.

**User Response:** See “Using topics” on page 221 for more information on Topic implementations.

---

**MQJMS6118** Topic {0} contains a wildcard, which is invalid for publishing.

**Explanation:** The Topic specified to a TopicPublisher method contained a wildcard. Wildcards are not allowed in Topics when publishing messages. The failing Topic is given by {0}.

**User Response:** See “Using topics” on page 221 for more information.

---

**MQJMS6119** An IOException occurred while publishing, exception {0}.

**Explanation:** An IOException was caught while publishing a message. {0} gives details of the exception.

**User Response:** See Chapter 11, “Writing WebSphere MQ JMS publish/subscribe applications,” on page 213 for more information.

---

**MQJMS6120** Attempted to use a temporary topic not created on the current connection.

**Explanation:** Invalid use of temporary topics and connections.

**User Response:** See Chapter 15, “JMS interfaces and classes,” on page 295 for more information.

---

**MQJMS6121** An IOException occurred while subscribing, exception {0}.

**Explanation:** An IOException was caught while subscribing. {0} gives details of the exception.

**User Response:** See Chapter 11, “Writing WebSphere MQ JMS publish/subscribe applications,” on page 213 for more information.

---

**MQJMS6122** An exception occurred when creating subscription to {0}, {1}.

**Explanation:** An invalid subject or query syntax was used in the creation of a subscriber, resulting in an exception. The topic name, and caught exception are included as parameters of this event.

**User Response:** See Chapter 11, “Writing WebSphere MQ JMS publish/subscribe applications,” on page 213 and the broker documentation for more information.

---

**MQJMS6232** While creating a TopicSubscriber, attempting to add the subscription to the matching engine resulted in exception {0}.

**Explanation:** {0} gives details of the exception.

**User Response:** See Chapter 11, “Writing WebSphere MQ JMS publish/subscribe applications,” on page 213 and the broker documentation for more information.

---

**MQJMS6234** An attempt was made to remove an object with Topic {0} from an empty matching engine: {1}.

**Explanation:** An attempt was made to remove from a null tree in match space. {0} gives the Topic and {1} gives the MatchTarget. Internal error.

**User Response:** Contact your IBM representative.

---

**MQJMS6235** An attempt was made to remove an object with a Topic {0} from the matching engine, but it did not have a cache entry: {1}.

**Explanation:** Internal error.

**User Response:** Contact your IBM representative.

---

**MQJMS6238** In attempting to access a field of a message, the following exception occurred: {0}.

**Explanation:** A corrupt message format was discovered. Internal error.

**User Response:** Contact your IBM representative.
MQJMS6240  An EvalCache get or put operation specified an invalid id.
Explanation:  An operation expected the MinValue of an EvalCache to be increased, but it won't be. Internal Error.
User Response:  Contact your IBM representative.

MQJMS6241  Too many content attributes were specified.
Explanation:  Too many non-topic attributes were specified in Factor.createMatcherInternal. Internal error.
User Response:  Contact your IBM representative.

MQJMS6246  An incorrect use of the Topic wildcard character {0} was detected.
Explanation:  The failing Topic is given by parameter {0}. 
User Response:  See "Using topics" on page 221 for more information.

MQJMS6247  The Topic segment separator {0} appears in an incorrect position.
Explanation:  A subscription Topic separator was used incorrectly. {0} shows the bad separator.
User Response:  See "Using topics" on page 221 for more information.

MQJMS6249  The following exception occurred while parsing a subscription selector: {0}.
Explanation:  A TypeCheckException occurred while loading or invoking the match parser. This may indicate a syntax error in your Selector.
User Response: For more information, see "Message selectors" on page 207. For information specific to JMS 1.1, see "Message selectors" on page 243.

MQJMS6250  The escape character was used to terminate the following pattern: {0}.
Explanation:  This may indicate a syntax error in your Selector.
User Response: For more information, see "Message selectors" on page 207. For information specific to JMS 1.1, see "Message selectors" on page 243.

MQJMS6251  The escape character {0} passed to the pattern tool is longer than one character.
Explanation:  This may indicate a syntax error in your Selector.
User Response: For more information, see "Message selectors" on page 207. For information specific to JMS 1.1, see "Message selectors" on page 243.
JMS exception messages
Appendix J. Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.
Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM’s application programming interfaces.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

- AIX
- IBM
- MQSeries
- OS/400
- System/390
- WebSphere
- AS/400
- iSeries
- MVS/ESA
- SecureWay
- S/390
- z/OS
- CICS
- Language Environment
- OS/390
- SupportPac
- VisualAge
- zSeries

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.
Java, HotJava, JDK, and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.
index

A
accessibility 24
JMS Postcard 24
accessing queues and processes 75
administered objects 47, 200
with WebSphere Application Server V4 475
administering JMS objects 45
administration commands 44
verbs 45
administration tool
configuration file 42
configuring 42
overview 41
properties 42
property mapping 457
starting 41
advantages of Java interface 63
applets
example code 69
running 93
security settings for 481
using WebSphere MQ Java in versus applications 67
appletviewer
using 5
application example 72
Application Server Facilities 277
classes and functions 277
sample client applications 287
sample code 283
applications
closing 209
JMS 1.1, writing 235
JMS publish/subscribe, writing 213
JMS, writing 199
running 94
unexpected termination 230
versus applets 67
ASF (Application Server Facilities) 277
ASFClient1.java 289
ASFClient2.java 290
ASFClient3.java 292
ASFClient4.java 293
ASFClient5.java 294
asynchronous message delivery 208
using JMS 1.1 248

B
bean-managed transactions 476
sample application 478
behavior in different environments 485
benefits of JMS 3
bindings
connection 6
connection, programming 68
example application 72
verifying 16
bindings transport, choosing 202
body; message 257
broker reports 233
BROKERCCDSUBQ object property 49, 279, 457
BROKERCCSUBQ object property 49, 279, 457
BROKERCONQ object property 49, 457
BROKERDURSUBQ object property 49, 457
BROKERPUBQ object property 49, 457
BROKERQMG object property 49, 457
BROKERSUBQ object property 49, 457
BROKERSERVER object property 49, 457
building a connection 200
using JMS 1.1 236
bytes message 257
BytesMessage
interface 300
type 207
CCSID object property 49, 457
certificate revocation list (CRL) 91
CHANGE (administration verb) 45
CHANNEL object property 49, 457
choosing transport 202
CICS Transaction Server
running applications 485
CipherSpecs 487
CipherSuites supported 487
class library 65
classes, Application Server Facilities 277
classes, core 95
restrictions and variations 96, 485
classes, JMS 295
classes, WebSphere MQ classes for Java 101
ManagedConnection 188
ManagedConnectionFactory 191
ManagedConnectionMetaData 193
MQC 179
MQChannelDefinition 102
MQChannelExit 104
MQConnectionManager 181
MQDistributionList 107
MQDistributionListItem 109
MQEnvironment 110
MQException 117
MQGetMessageOptions 119
MQManagedObject 123
MQMessage 126
MQMessageTracker 144
MQPoolServices 146
MQPoolServicesEvent 147
MQPoolServicesEventListener 180
MQPoolToken 149
MQProcess 150
MQPutMessageOptions 152
MQQueue 155
classes, WebSphere MQ classes for Java (continued)
MQQueueManager 163
MQReceiveExit 182
MQSecurityExit 184
MQSendExit 186
MQSimpleConnectionFactory 176
classpath
configuring 25
settings 10
Cleanup
class 308
CLEANUP object property 49, 457
cleanup utility
comsumer 248
subscriber 230
CLEANUPINT object property 49, 457
client properties 56
client transport, choosing 202
CLIENTID object property 49, 457
clients
configuring queue manager 15
connection 5
programming 67
verifying 16
closing
applications 209
JMS resources in publish/subscribe
mode 219
resources 208
resources using JMS 1.1 252
code examples 69
com.ibm.jms package 299
com.ibm.mq.jar 9
com.ibm.mq.jms package 298
com.ibm.mqbind.jar 9
com.ibm.mq.jms.jar 9
combinations, valid, of objects and properties 52
commands, administration 44
compiling WebSphere MQ classes for Java programs 93
configuration file, for administration tool 42
configuring
environment variables 26
for publish/subscribe 26
for WebSphere Application Server 44
Java 2 Security Manager 13
LDAP server 463
queue manager for clients 15
the administration tool 42
to run applets 481
unsupported
InitialContextFactory 43
Web server 12
your classpath 25
your installation 25
confirm on arrival report options, message 133
confirm on delivery report options, message 133
connecting to a publish/subscribe broker 469
connecting to a queue manager 75
connecting to WebSphere Business Integration Event Broker
  configuring a client for a multicast connection 473
  configuring a client for connection through a proxy server 473
  configuring a client for HTTP tunnelling 473
  configuring a client for SSL authentication 472
  configuring the broker for a direct connection 470
connecting to WebSphere Business Integration Message Broker
  configuring a client for a multicast connection 473
  configuring a client for connection through a proxy server 473
  configuring a client for HTTP tunnelling 473
  configuring a client for SSL authentication 472
  configuring the broker for a direct connection 470
connecting to WebSphere MQ Event Broker 469
connecting to WebSphere MQ Integrator V2 469
connection
  building 200
  building using JMS 1.1 236
  creating 201
  interface 199
  options 4
  starting 201
WebSphere MQ, losing 230
Connection interface 313
connection pooling 80
example 81
connection type, defining 68
ConnectionConsumer class 277
ConnectionConsumer interface 318
ConnectionFactory interface 319
ConnectionMetaData interface 335
cnct.jar 9
customer cleanup utility 248
customer-managed transactions 476
  sample application 477
converting the log file 41
COPY (administration verb) 45
core classes 95
  restrictions and variations 96, 485
createQueueSession method 203
createReceiver method 207
createSender method 204
creating
  a connection 201
  factories at runtime 201
JMS objects 48
  Topics at runtime 223

default connection pool 80
  multiple components 83
default trace and log output locations 38
DEFINE (administration verb) 45
  defining connection type 68
  defining transport 202
  definition, LDAP schema 463
DELETE (administration verb) 45
DeliveryMode interface 337
  dependencies, property 56
DESCRIPTION object property 49, 457
  Destination interface 338
  destinations 239
  differences between applets and applications 67
  differences due to environment 485
  DIRECTAUTH object property 49, 457
directories, installation 10
  disconnecting from a queue manager 75
DISPLAY (administration verb) 45
disposition options, message 134, 281
distribution lists
  platform dependency 98
  durable subscribers 224

ENCODING object property 57, 457
END (administration verb) 45
  environment dependencies 95
  functions not with all platforms 98
  distribution lists 98
  MQGetMessageOptions fields 98
  MQMD fields 99
  MQPutMessageOptions fields 98
  MQQueueManager begin() method 98
  MQQueueManager constructor 98
  restrictions and variations 96
  MQGMO_.* values 96
  MQPMO_.* values 96
  MQPMRF_.* values 96
  MQRO_.* values 97
  z/OS and OS/390 97
  environment differences 485
  environment variables 10
  configuring 26
error
  conditions when creating an object 59
  conditions when using an object 59
  handling 77
  logging 39
  recovery, IVT 34
  recovery, PSIVT 37
  runtime, handling 209
  runtime, handling using JMS 1.1 252
  error messages 18
  LDAP server 463
example code 69
exception listener 209
exception messages, JMS 489
exception report options, message 133, 281
ExceptionListener interface 340

exceptions
  JMS 209
  JMS 1.1 252
  WebSphere MQ 209
exit string properties 57
expiration report options, message 133
EXPIRY object property 49, 457
extra function provided over WebSphere MQ Java 3

factories, creating at runtime 201
FAILIFQUEESCE object property 49, 457
formatLog utility 41, 461
fscontext.jar 9
function, extra provided over WebSphere MQ Java 3
functions, Application Server Facilities 277

getting started 3

handling
  errors 77
  JMS runtime errors 209
  messages 76
  runtime errors using JMS 1.1 252
headers, message 257
HOSTNAME object property 49, 457

import statements 217
INITIAL_CONTEXT_FACTORY property 42, 43
inquire and set 78
installation
  directories 10
  Installation Verification Test program
    for publish/subscribe (PSIVT) 35
  IVT error recovery 34
  PSIVT error recovery 37
  setup 25
  verifying 19
Installation Verification Test program (IVT) 31
installing
  WebSphere MQ classes for Java 9
  WebSphere MQ classes for Java Message Service 9
interface, programming 64
interfaces
  JMS 199, 295
  WebSphere MQ 199
introduction
  for programmers 63
  WebSphere MQ classes for Java 3
  WebSphere MQ classes for Java Message Service 3
IVT (Installation Verification Test program) 31
IVTrun utility 461
IVTRun utility 31, 33, 37
IVTSetup utility 32, 461
IVTTidy utility 34, 461

**J**

J2EE connector architecture 81
JAAS (Java Authentication and Authorization Service) 81, 181
jar files 9
Java 2 Platform Enterprise Edition (J2EE) 81
Java 2 Security Manager, running applications under 13
Java Authentication and Authorization Service (JAAS) 81, 181
Java classes 65
See classes, WebSphere MQ classes for Java
Java Development Kit (JDK) 64
Java interface, advantages 63
Java Transaction API (JTA) 449
with WebSphere Application Server V4 475
javaClassName LDAP attribute setting 464
javaClassNames LDAP attribute setting 464
javaCodebase LDAP attribute setting 464
javaContainer LDAP attribute setting 464
javaFactory LDAP objectClass definition 466
javaNamingReference LDAP attribute setting 465
javaObject LDAP objectClass definition 466
javaReferenceAddress LDAP objectClass definition 466
javaSerializedData LDAP attribute setting 465
javaSerializedObject LDAP objectClass definition 465
javax.jms package 295
JDBC coordination 87
JDK (Java Development Kit) 64
JMS
administered objects 200
applications, writing 199
benefits 3
classes 295
exception listener 209
exceptions 209
reference 489
interfaces 199, 295
introduction 3
mapping of fields at send or publish 268
mapping with MQMD 265
message types 206
messages 257
model 199
objects for publish/subscribe 217
JMS (continued)
objects, administering 45
objects, creating 48
objects, properties 49
publish/subscribe applications, writing 213
resources, closing in publish/subscribe mode 219
JMS 1.1
applications, writing 235
exceptions 252
model 235
JMS exception messages 489
JMS JTA/XA Interface
with WebSphere Application Server V4 475
JMS Postcard accessibility 24
changing appearance 24
changing browser for help 24
default configuration 22
font and color settings 24
how it works 22
interoperability with other Postcard applications 24
receiving messages 23
sending a postcard 20
sending messages 23
sign-on 20
advanced options 20
starting 19
tidying up after use 24
using with one queue manager 20
using with two queue managers 21
jms.jar 9
JMSAdmin configuration file 42, 43
JMSAdmin utility 41, 461
JMSAdmin.config file 41
JMSBytesMessage class 300
JMSCorrelationID header field 257
JMSMapMessage class 341
JMSMessage class 349
JMSStreamMessage class 405
JMSTextMessage class 415
JNDI
retrieving 200
security considerations 43
jndi.jar 9
JSSE
for SSL support 89, 210, 253
JTA (Java Transaction API) 449
with WebSphere Application Server V4 475
JTA/JDBC coordination 87
installation
other platforms 87
Windows 87
known problems 88
limitations 88
switch file 87
usage 88
LDAP server (continued)
attribute settings
javaClassName 464
javaClassNames 464
javaCodebase 464
javaFactory 465
javaReferenceAddress 465
javaSerializedData 465
configuration 463
error messages 463
iSeries OS/400 V4R5 Schema Modification 467
Microsoft Active Directory 466
Netscape Directory 466
objectClass definitions
javaContainer 466
javaNamingReference 466
javaObject 466
javaSerializedObject 465
schema 463
Sun Microsystems’ Schema Modification Applications 467
ldap.jar 9
library, Java classes 65
listener, JMS exception 209
Load1.java 287
Load2.java 290
local publications, suppressing 225
LOCALADDRESS object property 49,
457
log file
converting 41
default output location 38
logging errors 39

**M**

MAIG, SupportPac special considerations for 483
ManagedConnection 188
ManagedConnectionFactory 191
ManagedConnectionMetaData 193
manipulating subcontexts 45
map message 257
MapMessage interface 341
type 207
mapping properties between admin. tool and programs 457
mcd folder 471
message
body 257
delivery, asynchronous 208
delivery, asynchronous using JMS 1.1 248
error 18
handling 76
headers 257
message body 273
properties 257
selectors 208, 257
selectors and SQL 258
selectors in publish/subscribe
mode 224
types 206, 257
Message interface 349
MessageConsumer interface 199, 363

Index 511
null
programming (continued)
  tracing 94
  writing 67
programming interface 64
programs
  JMS 1.1, writing 235
  JMS publish/subscribe, writing 213
  JMS, writing 199
  running 38, 94
  tracing 38
properties
  client 56
  dependencies 56
  for Secure Sockets Layer 58
  for WebSphere MQ Event Broker 57
  mapping between admin. tool and programs 457
message 257
  of exit strings 57
  of JMS objects 49
  queue, setting 204
properties and objects, valid combinations 52
PROVIDER_PASSWORD property 43
PROVIDER_URL property 42
PROVIDER_USERDN property 43
providerutil.jar 9
PROXYHOSTNAME object property 49, 457
PROXYPORT object property 49, 457
PSIVT (Installation Verification Test program) 35
PSIVTRun utility 35, 461
PSReportDump application 233
PUBACKINT object property 49, 457
publications (publish/subscribe), local suppressing 225
publish/subscribe
  installation verification test program (PSIVT) 35
  sample application with WebSphere Application Server V4 478
  setup for 26
  publish/subscribe broker, connecting to 469
  publishing messages 219
Queue
  QMANAGER object property 49, 457
  Queue interface 375
  object 200
queue manager
  configuring for clients 15
  connecting to 75
  disconnecting from 75
  operations on 74
QUEUE object property 49, 457
queue properties
  setting 204
  setting with set methods 206
QueueBrowser interface 377
QueueConnection interface 379
QueueReceiver interface 384
QueueRequestor class 385
queues, accessing 75
QueueSender interface 387
QueueSession interface 390
R
range of ports, specifying for client connections
  WebSphere MQ base Java 68
  WebSphere MQ JMS 203
  WebSphere MQ JMS 1.1 237
reading strings 77
receiving
  messages 207
  messages in publish/subscribe mode 219
  messages using JMS 1.1 241
  RECEEXIT object property 49, 457
  RECEEXITINIT object property 49, 457
  report options, message 133, 281
  reports, broker 233
resources
  closing 208
  closing using JMS 1.1 252
  restrictions and variations to core classes 485
  retrieving objects from JNDI 200
runjms utility 38, 461
running
  applets 93
  applications under CICS Transaction Server 485
  in a Web browser 5
  programs 38
  standalone program 5
  the IVT 31
  the PSIVT 35
  WebSphere MQ classes for Java applications 94
  with appletviewer 5
  your own programs 17
runtime
  creating factories 201
  creating Topics 223
  errors, handling 209
  errors, handling using JMS 1.1 252
Sample
  sample code (continued)
    ServerSession 283
    ServerSessionPool 283
  Sample1EJB.java 477
  Sample2EJB.java 478
  Sample3EJB.java 478
  schema definition, LDAP 463
  schema, LDAP server 463
  scripts provided with WebSphere MQ classes for Java Message Service 461
  SECEXIT object property 49, 457
  SECEXITINIT object property 49, 457
  Secure Sockets Layer 79, 210
  certificate revocation list (CRL) 91
  CipherSpecs 90, 487
  CipherSuites 90
  CipherSuites supported 487
  distinguished names (DN) 90
  enabling 90
  handled by JSSE 89, 210, 253
  introduction 89, 210, 253
  properties 58
  SSL.CERTSTORES 211, 255
  SSL.CIPHERSUITE 210, 254
  SSL.FINGERNAME 210, 254
  sslCertStores property 92
  sslCipherSuite property 90
  sslPeerName property 90
  sslSocketFactory property 92
  using JMS 1.1 253
  with user exits 79
security considerations, JNDI 43
Security policy definition file, editing 13
SECURITY_AUTHENTICATION property 42, 43
selecting a subset of messages 208, 257
selectors
  message 208, 257
  message in publish/subscribe mode 224
  message, and SQL 258
SENDEXIT object property 49, 457
SENDEXITINIT object property 49, 457
sending
  messages 204
  messages using JMS 1.1 240
ServerSession sample code 283
ServerSessionPool sample code 283
session
  obtaining 203
  obtaining using JMS 1.1 238
Session class 277
Session interface 199, 393
set and inquire 78
set methods
  on MQQueueConnectionFactory 202
  using to set queue properties 206
setJMSType method 471
setting
  queue properties 204
  queue properties with set methods 206
shutting down applications 209
software, prerequisites 6
solving problems 17
  general 38
  in publish/subscribe mode 229
Index 513
SPARSESUBS object property 49, 457
SQL for message selectors 258
SSL
   See Secure Sockets Layer
SQLCERTSTORES object property 211, 235
sslCertStores property 92
SSLCIPHERSUITE object property 49, 58, 210, 254, 457
sslCipherSuite property 90
SSLRL object property 49, 58, 457
SSLPEERNAME object property 49, 58, 210, 254, 457
sslPeerName property 90
sslSocketFactory property 92
standalone program, running 5
starting a connection 201
starting the administration tool 41
STATREFRESHINT object property 49, 457
stream message 257
StreamMessage interface 405
type 207
strings, reading and writing 77
subcontexts, manipulating 45
subscriber cleanup utility 230
subscriber options 224
subscriptions, receiving 219
subset of messages, selecting 208, 257
SUBSTORE object property 49
Sun JMS interfaces and classes 295
Sun Web site 3
SupportPac MAIG
   special considerations for 483
suppressing local publications 225
switch file for JTA/JDBC 87
SYNCPOINTALLGETS object property 49, 457

T
TARGCLIENT object property 49, 457
TCP/IP
   client verifying 16
   connection, programming 67
TEMPMODEL object property 49, 457
TemporaryQueue interface 413
TemporaryTopic interface 414
TEMPQPREFIX object property 49
termination, unexpected 230
testing WebSphere MQ classes for Java programs 94
text message 257
TextMessage interface 415
type 207
tokens, connection pooling 80
Topic
   interface 217, 416
   names 221
   names, wildcards 221
   object 200
TOPIC object property 49, 457
TopicConnection 217
interface 420
TopicConnectionFactory (continued)
   interface 423
TopicLoadJava 291
TopicPublisher 219
interface 431
TopicRequestor class 434
TopicSession 217
interface 436
TopicSubscriber 219
interface 440
trace, default output location 38
tracing
   programs 94
   the sample application 17
WebSphere MQ for Java Message Service 38
transactions
   bean-managed 476
   container-managed 476
   sample application 477, 478
TRANSPORT object property 49, 457
transport, choosing 202
two-phase commit
   with WebSphere Application Server
   V4 476
types of JMS message 206, 257

U
unexpected application termination 230
uniform resource identifier (URI) for queue properties 204
URI for queue properties 204
USE_INIT_DIR_CONTEXT property 43
USECONNPOLLING object property 457
USECONNPOLLING object property 49
user exits
   with SSL 79
   writing 79, 209
writing using JMS 1.1 253
uses for WebSphere MQ 4
using
   WebSphere MQ base Java 15
   utilities provided with WebSphere MQ classes for Java Message Service 461

V
valid combinations of objects and properties 52
verbs, WebSphere MQ supported 64
verification
   with JNDI (point-to-point) 32
   with JNDI (publish/subscribe) 36
   without JNDI (point-to-point) 31
   without JNDI (publish/subscribe) 35
verifying
   client mode installation 29
   TCP/IP clients 16
   with the sample appllet 29
   with the sample application 16
   your installation 19
   versions of software required 6
VisiBroker
   using 4

W
Web browser
   using 5
Web server, configuring 12
WebSphere Application Server 283
   configuration 44
   CosNaming namespace 42
   CosNaming repository 42, 44
WebSphere Application Server V4
   JMS JTA/XA Interface 475
   using with JMS 475
WebSphere Business Integration Event Broker, connecting to
   configuring a client for a multicast connection 473
   configuring a client for connection through a proxy server 473
   configuring a client for HTTP tunnelling 473
   configuring a client for SSL authentication 472
   configuring the broker for a direct connection 470
WebSphere Business Integration Message Broker, connecting to
   configuring a client for a multicast connection 473
   configuring a client for connection through a proxy server 473
   configuring a client for HTTP tunnelling 473
   configuring a client for SSL authentication 472
   configuring the broker for a direct connection 470
WebSphere MQ
   connection, losing 230
   exceptions 209
   interfaces 199
   messages 261
WebSphere MQ classes for Java
   classes 101
WebSphere MQ Event Broker
   connecting as publish/subscribe broker 469
WebSphere MQ Event Broker
   properties 57
WebSphere MQ Integrator V2
   connecting as publish/subscribe broker 469
   transforming and routing messages 471
WebSphere MQ Message Descriptor (MQMD) 261
   mapping with JMS 265
WebSphere MQ Publish/Subscribe 26
WebSphere MQ supported verbs 64
wildcards in topic names 221
writing
   JMS 1.1 applications 235
   JMS applications 199
   JMS publish/subscribe applications 213
writing (continued)
  programs 67
  strings 77
  user exits 79, 209
  user exits using JMS 1.1 253

X
XAConnection interface 441
XAConnectionFactory interface 443
XAQueueConnection interface 379, 445
XAQueueConnectionFactory interface 381, 446
XAQueueSession interface 448
XAResource 449
XASession interface 449
XATopicConnection interface 451
XATopicConnectionFactory interface 452
XATopicSession interface 454

Z
z/OS and OS/390
  differences with 97
Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To make comments about the functions of IBM products or systems, talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

• By mail, to this address:
  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom

• By fax:
  – From outside the U.K., after your international access code use 44–1962–816151
  – From within the U.K., use 01962–816151

• Electronically, use the appropriate network ID:
  – IBM Mail Exchange: GBIBM2Q9 at IBMMMAIL
  – IBMLink™: HURSLEY(IDRCF)
  – Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

• The publication title and order number
• The topic to which your comment applies
• Your name and address/telephone number/fax number/network ID.
Spine information:

IBM

WebSphere MQ

Using Java