

MQSeries[®] Everyplace for Multiplatforms



Programming Reference

Version 1.1

MQSeries[®] Everyplace for Multiplatforms



Programming Reference

Version 1.1

Take Note!

Before using this information and the product it supports, be sure to read the general information under “Appendix. Notices” on page 397

Licence warning

MQSeries Everyplace Version 1.1 is a toolkit that enables users to write MQSeries Everyplace applications and to create an environment in which to run them.

The licence conditions under which the toolkit is purchased determine the environment in which it can be used:

*If MQSeries Everyplace is purchased for use as a **device** (client) it may **not** be used to create an **MQSeries Everyplace channel manager**, or an **MQSeries Everyplace channel listener**, or an **MQSeries Everyplace bridge***

*The presence of an **MQSeries Everyplace channel manager**, or an **MQSeries Everyplace channel listener**, or an **MQSeries Everyplace bridge** defines a **gateway** (server) environment, which requires a gateway licence.*

Second Edition (October 2000)

This edition applies to MQSeries Everyplace for Multiplatforms Version 1.1 and to all subsequent releases and modifications until otherwise indicated in new editions.

This document is continually being updated with new and improved information. For the latest edition, please see the MQSeries family library Web page at <http://www.ibm.com/software/ts/mqseries/library/>.

© Copyright International Business Machines Corporation 2000. All rights reserved.

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

Contents

About this book	xi
Who should read this book	xi
Prerequisite knowledge	xi
Terms used in this book	xi

Summary of Changes	xiii
Changes for this edition (SC34-5846-01).	xiii

Part 1. MQSeries Everyplace Java API 1

Chapter 1. MQSeries Everyplace classes and interfaces 3

com.ibm.mqe	3
com.ibm.mqe.administration	4
com.ibm.mqe.attributes	4
com.ibm.mqe.registry	5
com.ibm.mqe.server	5
com.ibm.mqe.mqemqmessage.	5
com.ibm.mqe.mqbridge.	6
com.ibm.mqe.adapters	6

Chapter 2. Classes in com.ibm.mqe. 9

MQe.	11
Constants	11
Publically accessible variables	14
Method summary	15
MQe abbreviate	15
MQe alias	16
MQe asciiToByte.	17
MQe byteToAscii	17
MQe byteToHex	17
MQe byteToInt	18
MQe byteToLong	18
MQe byteToShort	19
MQe byteToUnicode	19
MQe debug	19
MQe getEventLogHandler	20
MQe getTraceHandler	20
MQe hexToAscii	21
MQe hexToByte	21
MQe intToByte	22
MQe log	22
MQe mapFileDescriptor	23
MQe setEventLogHandler	23
MQe setTraceHandler	24
MQe sliceByteArray	24
MQe trace	25
MQe type	26
MQe unicodeToByte	26
MQe unicodeToUTF	27
MQe uniqueValue	27
MQe utfToUnicode	27
MQeAdapter	29

Method summary	29
MQeAdapter activate	29
MQeAdapter checkOption	30
MQeAdapter close	30
MQeAdapter control	31
MQeAdapter equals	31
MQeAdapter erase	31
MQeAdapter open	32
MQeAdapter qualityOfService	32
MQeAdapter read	32
MQeAdapter readEOF.	33
MQeAdapter readln	33
MQeAdapter readObject	34
MQeAdapter status.	34
MQeAdapter write	35
MQeAdapter writeln	35
MQeAdapter writeObject.	36
MQeAdminMsg	37
Constants and variables	37
Constructor summary	38
Method summary	38
MQeAdminMsg	39
MQeAdminMsg characteristics	39
MQeAdminMsg create.	40
MQeAdminMsg delete	40
MQeAdminMsg duplicate	41
MQeAdminMsg getAction	42
MQeAdminMsg getErrorFields	42
MQeAdminMsg getFieldInError	43
MQeAdminMsg getInputFields.	43
MQeAdminMsg getMaxAttempts	44
MQeAdminMsg getName	44
MQeAdminMsg getOutputFields	45
MQeAdminMsg getRC	45
MQeAdminMsg getReason	46
MQeAdminMsg getTargetQMgr	46
MQeAdminMsg inquire	47
MQeAdminMsg inquireAll	47
MQeAdminMsg setAction	48
MQeAdminMsg setName.	48
MQeAdminMsg setTargetQMgr	49
MQeAdminMsg update	49
MQeAttribute	51
Constructor summary	51
Method summary	51
MQeAttribute	51
MQeAttribute activate.	52
MQeAttribute authenticatedID	53
MQeAttribute change	53
MQeAttribute close.	53
MQeAttribute decodeData	54
MQeAttribute encodeData	54
MQeAttribute equals	55
MQeAttribute getAuthenticator.	55
MQeAttribute getCompressor	55
MQeAttribute getCryptor.	56

MQeChannelListener	57	MQeFields getIntArray	92
Constructor summary	57	MQeFields getLong	92
Method summary	57	MQeFields getLongArray	93
MQeChannelListener	57	MQeFields getShort	93
MQeChannelListener activate	58	MQeFields getShortArray	94
MQeChannelListener setTimer	59	MQeFields getUnicode	94
MQeChannelListener stop	59	MQeFields getUnicodeArray	95
MQeChannelManager	61	MQeFields hide	96
Constructor summary	61	MQeFields putArrayLength	96
Method summary	61	MQeFields putArrayOfByte	97
MQeChannelManager	61	MQeFields putArrayOfDouble	97
MQeChannelManager getGlobalHashtable	62	MQeFields putArrayOfFloat	98
MQeChannelManager mapDestination	62	MQeFields putArrayOfInt	98
MQeChannelManager numberOfChannels	63	MQeFields putArrayOfLong	99
MQeChannelManager process	63	MQeFields putArrayOfShort	99
MQeChannelManager timeOut	64	MQeFields putAscii	100
MQeChannelManager totalNumberOfChannels	64	MQeFields putAsciiArray	100
MQeEnumeration	66	MQeFields putBoolean	101
Method summary	66	MQeFields putByte	101
MQeEnumeration getLockId	66	MQeFields putByteArray	101
MQeEnumeration getNextMessage	66	MQeFields putDouble	102
MQeEnumeration getQueueManagerName	68	MQeFields putDoubleArray	102
MQeEnumeration getQueueName	68	MQeFields putFields	103
MQeException	70	MQeFields putFieldsArray	104
Constructor summary	70	MQeFields putFloat	104
Method summary	70	MQeFields putFloatArray	105
MQeException	70	MQeFields putInt	105
MQeException code	71	MQeFields putIntArray	106
MQeFields	72	MQeFields putLong	106
Constructor summary	72	MQeFields putLongArray	107
Method summary	72	MQeFields putShort	107
MQeFields	74	MQeFields putShortArray	108
MQeFields contains	74	MQeFields putUnicode	108
MQeFields copy	75	MQeFields putUnicodeArray	109
MQeFields dataType	76	MQeFields rename	109
MQeFields delete	76	MQeFields restore	110
MQeFields dump	77	MQeFields restoreFromFile	110
MQeFields dumpToString	79	MQeFields restoreFromString	111
MQeFields dumpedType	80	MQeFields setAttribute	112
MQeFields equals	80	MQeFields updateValue	113
MQeFields fields	81	MQeKey	114
MQeFields getArrayLength	81	Constructor summary	114
MQeFields getArrayOfByte	82	Method summary	114
MQeFields getArrayOfDouble	82	MQeKey	114
MQeFields getArrayOfFloat	83	MQeKey setLocalKey	114
MQeFields getArrayOfInt	84	MQeMessageEvent	116
MQeFields getArrayOfLong	84	Method summary	116
MQeFields getArrayOfShort	85	MQeMessageEvent getMsgFields	116
MQeFields getAscii	85	MQeMessageEvent getQueueManagerName	117
MQeFields getAsciiArray	86	MQeMessageEvent getQueueName	117
MQeFields getAttribute	86	MQeMsgObject class	119
MQeFields getBoolean	87	Constants and variables	119
MQeFields getByte	87	Constructor summary	120
MQeFields getByteArray	88	Method summary	120
MQeFields getDouble	88	MQeMsgObject	120
MQeFields getDoubleArray	89	MQeMsgObject getMsgUIDFields	121
MQeFields getFields	89	MQeMsgObject getOriginQMgr	122
MQeFields getFieldsArray	90	MQeMsgObject getTimeStamp	122
MQeFields getFloat	90	MQeMsgObject putOriginQMgr	122
MQeFields getFloatArray	91	MQeMsgObject resetMsgUIDFields	123
MQeFields getInt	91	MQeMsgObject unwrapMessageObject	123

MQeQueue	125	MQeQueueManagerConfigure	
Method summary	125	deleteSystemQueueDefinition	167
MQeQueue getCreationDate	126	MQeQueueManagerConfigure	
MQeQueue getDefaultPriority	127	queueManagerExists	168
MQeQueue getDescription	127	MQeQueueManagerConfigure	
MQeQueue getExpiryInterval	128	setChannelTimeout	168
MQeQueue getMaxMessageSize	128	MQeQueueManagerConfigure	
MQeQueue getMaxQueueSize	129	setChnlAttributeRuleName	169
MQeQueue getNumberOfMessages	129	MQeQueueManagerConfigure setDescription	170
MQeQueue getQueueAttribute	130	MQeQueueManagerRule	171
MQeQueue getQueueManagerName	131	Method summary	171
MQeQueue getQueueName	131	MQeQueueManagerRule activateQueues	171
MQeQueue getQueueStore	132	MQeQueueManagerRule addQueue	172
MQeQueueManager	134	MQeQueueManagerRule deleteMessage	173
Constructor summary	134	MQeQueueManagerRule getMessage	174
Method summary	134	MQeQueueManagerRule getRetryCount	175
MQeQueueManager	135	MQeQueueManagerRule peerConnection	176
MQeQueueManager activate	137	MQeQueueManagerRule putMessage	176
MQeQueueManager addMessageListener	138	MQeQueueManagerRule queueManagerActivate	177
MQeQueueManager browseMessages	139	MQeQueueManagerRule queueManagerClose	178
MQeQueueManager browseMessagesAndLock	141	MQeQueueManagerRule removeQueue	179
MQeQueueManager checkActive	143	MQeQueueManagerRule transmit	180
MQeQueueManager close	143	MQeQueueManagerRule triggerTransmission	181
MQeQueueManager confirmGetMessage	144	MQeQueueManagerRule undo	183
MQeQueueManager confirmPutMessage	145	MQeQueueRule	184
MQeQueueManager deleteMessage	146	Method summary	185
MQeQueueManager getMessage	147	MQeQueueRule addListener	185
MQeQueueManager getName	149	MQeQueueRule attributeChange	187
MQeQueueManager getReference	150	MQeQueueRule browseMessage	187
MQeQueueManager putMessage	150	MQeQueueRule deleteMessage	188
MQeQueueManager removeMessageListener	152	MQeQueueRule duplicateMessage	189
MQeQueueManager triggerTransmission	153	MQeQueueRule filterMessage	190
MQeQueueManager undo	154	MQeQueueRule getMessage	190
MQeQueueManager unlockMessage	155	MQeQueueRule getPendingMessage	191
MQeQueueManager waitForMessage	156	MQeQueueRule indexEntry	192
MQeQueueManagerConfigure	158	MQeQueueRule messageExpired	193
Constructor summary	158	MQeQueueRule putMessage	194
Method summary	158	MQeQueueRule queueActivate	194
MQeQueueManagerConfigure	158	MQeQueueRule queueClose	195
MQeQueueManagerConfigure activate	159	MQeQueueRule removeListener	196
MQeQueueManagerConfigure close	160	MQeQueueRule resetMessageLock	197
MQeQueueManagerConfigure		MQeQueueRule undo	198
defineDefaultAdminQueue	161	MQeQueueRule useCountChanged	199
MQeQueueManagerConfigure		MQeEventLogInterface	200
defineDefaultAdminReplyQueue	161	Method summary	200
MQeQueueManagerConfigure		MQeEventLogInterface activate	200
defineDefaultDeadLetterQueue	162	MQeEventLogInterface close	200
MQeQueueManagerConfigure		MQeEventLogInterface logOutput	200
defineDefaultSystemQueue	163	MQeMessageListenerInterface	201
MQeQueueManagerConfigure		Method summary	201
defineQueueManager	163	MQeMessageListener messageArrived	201
MQeQueueManagerConfigure		MQeRunListInterface	202
deleteAdminQueueDefinition	164	Method summary	202
MQeQueueManagerConfigure		MQeRunListInterface activate	202
deleteAdminReplyQueueDefinition	164	MQeSecurityInterface	205
MQeQueueManagerConfigure		Method summary	205
deleteDeadLetterQueueDefinition	165	MQeSecurityInterface alias	205
MQeQueueManagerConfigure		MQeSecurityInterface channelCommand	205
deleteQueueManagerDefinition	166	MQeSecurityInterface newAdapter	206
MQeQueueManagerConfigure		MQeSecurityInterface mapFileDescriptor	206
deleteStandardQMDefinitions	166	MQeTraceInterface	207

Method summary	207
MQeTraceInterface activate	207
MQeTraceInterface traceMessage	207
MQeTraceInterface addMessage	208
MQeTraceInterface addMessageBundle	208
Standard trace messages.	209
MQeTraceInterface close	211
MQeTraceInterface getMessage	211

Chapter 3. Classes in com.ibm.mqe.administration 213

MQeAdminQueueAdminMsg	214
Constants and variables	214
MQeConnectionAdminMsg.	215
Constants and variables	215
Method summary	216
MQeConnectionAdminMsg create	217
MQeConnectionAdminMsg update	217
MQeHomeServerQueueAdminMsg	219
Constants and variables	219
MQeQueueAdminMsg	220
Constants and variables	220
Constructor summary	222
Method summary	223
MQeQueueAdminMsg	223
MQeQueueAdminMsg addAlias	223
MQeQueueAdminMsg removeAlias	224
MQeQueueAdminMsg setName	224
MQeQueueManagerAdminMsg	226
Constants and variables	226
MQeRemoteQueueAdminMsg.	228
Constants and variables	228
MQeStoreAndForwardQueueAdminMsg	229
Constants and variables	229
Method summary	229
MQeStoreAndForwardQueueAdminMsg addQueueManager	229
MQeStoreAndForwardQueueAdminMsg removeQueueManager	230

Chapter 4. Classes in com.ibm.mqe.attributes 231

MQe3DESCryptor	232
MQe3DESCryptor	232
MQeDESCryptor	233
MQeDESCryptor	233
MQeGenDH.	234
Method summary	234
MQeGenDH main	234
MQeGenDH genParams	235
MQeLocalSecure	236
Constructor summary	236
Method summary	236
MQeLocalSecure	236
MQeLocalSecure open	236
MQeLocalSecure read	237
MQeLocalSecure write	237
MQeLZWCompressor	239
MQeLZWCompressor	239
MQeMARSCryptor	240

MQeMARSCryptor	240
MQeMAttribute	241
Constructor summary	241
Method summary	241
MQeMAttribute	241
MQeMAttribute decodeData	242
MQeMAttribute encodeData	243
MQeMTrustAttribute	244
Constructor summary	244
Method summary	244
MQeMTrustAttribute	244
MQeMTrustAttribute decodeData	246
MQeMTrustAttribute encodeData.	247
MQeMTrustAttribute setHomeServer	247
MQeMTrustAttribute setPrivateRegistry	248
MQeMTrustAttribute setPublicRegistry	248
MQeRC4Cryptor	249
MQeRC4Cryptor	249
MQeRC6Cryptor	250
MQeRC6Cryptor	250
MQeRleCompressor	251
MQeRleCompressor	251
MQeWTLSCertAuthenticator	252
MQeWTLSCertAuthenticator	252
MQeXORCryptor	253
Constructor summary	253
Method summary	253
MQeXORCryptor	253
MQeXORCryptor setDecryptKey	254
MQeXORCryptor setEncryptKey	254

Chapter 5. Classes in com.ibm.mqe.registry 255

MQePrivateRegistry	256
Constructor summary	256
Method summary	256
MQePrivateRegistry	256
MQePrivateRegistry activate	256
MQePrivateRegistry deleteCertificate	258
MQePrivateRegistry getCertificate	258
MQePrivateRegistry getRegistryName	259
MQePrivateRegistry resetPIN	259
MQePrivateRegistry setTargetRegistryName	259
MQePrivateRegistryConfigure	261
Constructor summary	261
Method summary	261
MQePrivateRegistryConfigure	261
MQePrivateRegistryConfigure activate	262
MQePrivateRegistryConfigure close	264
MQePrivateRegistryConfigure credentialsExist	264
MQePrivateRegistryConfigure getCredentials	265
MQePrivateRegistryConfigure isPrivate.	267
MQePublicRegistry	268
Constructor summary	268
Method summary	268
MQePublicRegistry	268
MQePublicRegistry activate	268
MQePublicRegistry deleteCertificate.	269
MQePublicRegistry getCertificate.	270
MQePublicRegistry putCertificate	270
MQePublicRegistry requestCertificate	270

MQePublicRegistry shareCertificate 271

Chapter 6. Classes in **com.ibm.mqe.server** 273

MQeMiniCertIssuanceInterface	274
Constants	274
Method summary	274
MQeMiniCertIssuanceInterface addAuthenticatableEntity	274
MQeMiniCertIssuanceInterface addEntityRegisteredAddress	275
MQeMiniCertIssuanceInterface authoriseMiniCertRequest	275
MQeMiniCertIssuanceInterface deleteAuthenticatableEntity	276
MQeMiniCertIssuanceInterface deleteEntityRegisteredAddress	276
MQeMiniCertIssuanceInterface readAuthenticatableEntity	276
MQeMiniCertIssuanceInterface readEntityRegisteredAddress	277
MQeMiniCertIssuanceInterface SetRegistry	277
MQeMiniCertIssuanceInterface updateAuthenticatableEntity	277
MQeMiniCertIssuanceInterface updateEntityRegisteredAddress	278

Chapter 7. Classes in **com.ibm.mqe.mqemqmessage** 279

MQeMQMsgObject class	280
Constructor summary	280
Method summary	280
MQeMQMsgObject	282
MQeMQMsgObject dumpAllToString	282
MQeMQMsgObject dumpToString	283
MQeMQMsgObject equals	283
MQeMQMsgObject getAccountingToken	283
MQeMQMsgObject getApplicationIdData	284
MQeMQMsgObject getApplicationOriginData	285
MQeMQMsgObject getBackoutCount	285
MQeMQMsgObject getCharacterSet	286
MQeMQMsgObject getData	286
MQeMQMsgObject getCorrelationId	287
MQeMQMsgObject getEncoding	287
MQeMQMsgObject getExpiry	288
MQeMQMsgObject getFeedback	288
MQeMQMsgObject getFormat	289
MQeMQMsgObject getGroupId	289
MQeMQMsgObject getMessageFlags	290
MQeMQMsgObject getMessageId	290
MQeMQMsgObject getMessageSequenceNumber	291
MQeMQMsgObject getMessageType	291
MQeMQMsgObject getOffset	292
MQeMQMsgObject getOriginalLength	293
MQeMQMsgObject getPersistence	293
MQeMQMsgObject getPriority	294
MQeMQMsgObject getPutApplicationName	294
MQeMQMsgObject getPutApplicationType	295
MQeMQMsgObject getPutDateTime	295

MQeMQMsgObject getReplyToQueueManagerName	296
MQeMQMsgObject getReplyToQueueName	296
MQeMQMsgObject getReport	297
MQeMQMsgObject getUserId	297
MQeMQMsgObject setAccountingToken	298
MQeMQMsgObject setApplicationIdData	298
MQeMQMsgObject setApplicationOriginData	299
MQeMQMsgObject setBackoutCount	300
MQeMQMsgObject setCharacterSet	300
MQeMQMsgObject setCorrelationId	301
MQeMQMsgObject setData	301
MQeMQMsgObject setEncoding	302
MQeMQMsgObject setExpiry	302
MQeMQMsgObject setFeedback	303
MQeMQMsgObject setFormat	303
MQeMQMsgObject setGroupId	304
MQeMQMsgObject setMessageFlags	304
MQeMQMsgObject setMessageId	305
MQeMQMsgObject setMessageSequenceNumber	305
MQeMQMsgObject setMessageType	306
MQeMQMsgObject setOffset	307
MQeMQMsgObject setOriginalLength	307
MQeMQMsgObject setPersistence	308
MQeMQMsgObject setPriority	308
MQeMQMsgObject setPutApplicationName	309
MQeMQMsgObject setPutApplicationType	309
MQeMQMsgObject setPutDateTime	310
MQeMQMsgObject setReplyToQueueManagerName	310
MQeMQMsgObject setReplyToQueueName	311
MQeMQMsgObject setReport	311
MQeMQMsgObject setUserId	312

Chapter 8. Classes in **com.ibm.mqe.mqbridge** 313

MQeCharacteristicLabels	314
Constants and variables	314
MQeCharacteristicLabels	321
MQeClientConnectionAdminMsg	322
Constants and variables	322
Constructor summary	322
Method summary	322
MQeClientConnectionAdminMsg	322
MQeClientConnectionAdminMsg characteristics	323
MQeClientConnectionAdminMsg getClientConnectionName	324
MQeClientConnectionAdminMsg getName	324
MQeClientConnectionAdminMsg putClientConnectionName	325
MQeClientConnectionAdminMsg setName	325
MQeListenerAdminMsg	327
Constructor summary	327
Method summary	327
MQeListenerAdminMsg	327
MQeListenerAdminMsg characteristics	328
MQeListenerAdminMsg getListenerName	329
MQeListenerAdminMsg getName	329
MQeListenerAdminMsg putListenerName	330
MQeListenerAdminMsg setName	330
MQeMQBridgeAdminMsg	332

Constants and variables	332
Constructor summary	332
Method summary	332
MQeMQBridgeAdminMsg	332
MQeMQBridgeAdminMsg characteristics	333
MQeMQBridgeAdminMsg create	333
MQeMQBridgeAdminMsg getBridgeName	334
MQeMQBridgeAdminMsg getName	334
MQeMQBridgeAdminMsg delete	335
MQeMQBridgeAdminMsg putBridgeName	336
MQeMQBridgeAdminMsg setName	336
MQeMQBridgeQueue	337
Method summary	337
MQeMQBridgeQueue getMQMGr	337
MQeMQBridgeQueue getQueueAttribute	338
MQeMQBridgeQueue getQueueManagerName	338
MQeMQBridgeQueue getQueueName	338
MQeMQBridgeQueue getRemoteQName	339
MQeMQBridgeQueueAdminMsg	340
Constants and variables	340
Constructor summary	344
Method summary	344
MQeMQBridgeQueueAdminMsg	344
MQeMQBridgeQueueAdminMsg characteristics	345
MQeMQBridges	346
Constructor summary	346
Method summary	346
MQeMQBridges	346
MQeMQBridges activate	347
MQeMQBridges close	347
MQeMQBridgesAdminMsg	349
Constants and variables	349
Constructor summary	349
Method summary	349
MQeMQBridgesAdminMsg	349
MQeMQBridgesAdminMsg characteristics	350
MQeMQBridgesAdminMsg getName	350
MQeMQBridgesAdminMsg start	351
MQeMQBridgesAdminMsg stop	352
MQeMQMGrProxyAdminMsg	354
Constructor summary	354
Method summary	354
MQeMQMGrProxyAdminMsg	354
MQeMQMGrProxyAdminMsg characteristics	355
MQeMQMGrProxyAdminMsg	355
getMQMGrProxyName	355
MQeMQMGrProxyAdminMsg getName	356
MQeMQMGrProxyAdminMsg	356
putMQMGrProxyName	356
MQeMQMGrProxyAdminMsg setName	357
MQeRunState	358
Constants and variables	358
MQeTransformerInterface	359
Method summary	359
MQeTransformerInterface activate	359
MQeTransformerInterface transform	359

Chapter 9. Classes in	
com.ibm.mqe.adapters	361
MQeDiskFieldsAdapter	362
Method summary	362

MQeDiskFieldsAdapter accept	362
MQeDiskFieldsAdapter activate	362
MQeDiskFieldsAdapter close	363
MQeDiskFieldsAdapter control	364
MQeDiskFieldsAdapter erase	364
MQeDiskFieldsAdapter open	365
MQeDiskFieldsAdapter readObject	365
MQeDiskFieldsAdapter status	365
MQeDiskFieldsAdapter writeObject	366
MQeMemoryFieldsAdapter	367
Method summary	367
MQeMemoryFieldsAdapter activate	367
MQeMemoryFieldsAdapter close	368
MQeMemoryFieldsAdapter control	368
MQeMemoryFieldsAdapter erase	369
MQeMemoryFieldsAdapter open	369
MQeMemoryFieldsAdapter readObject	369
MQeMemoryFieldsAdapter status	370
MQeMemoryFieldsAdapter writeObject	370
MQeTcpipAdapter	371
Method summary	371
MQeTcpipAdapter activate	371
MQeTcpipAdapter close	372
MQeTcpipAdapter control	373
MQeTcpipAdapter open	373
MQeTcpipAdapter read	374
MQeTcpipAdapter readln	374
MQeTcpipAdapter status	375
MQeTcpipAdapter write	375
MQeTcpipAdapter writeln	375
MQeTcpipHttpAdapter	377
Method summary	377
MQeTcpipHttpAdapter read	377
MQeTcpipHttpAdapter readEOF	378
MQeTcpipHttpAdapter readln	378
MQeTcpipHttpAdapter write	379
MQeTcpipHttpAdapter writeln	379
MQeTcpipLengthAdapter	381
Method summary	381
MQeTcpipLengthAdapter read	381
MQeTcpipLengthAdapter write	381
MQeTcpipHistoryAdapter	383
Method summary	383
MQeTcpipHistoryAdapter activate	383
MQeTcpipHistoryAdapter close	384

Part 2. Exceptions and return codes 385

Chapter 10. Exceptions	387
Ordered numerically	387
Ordered alphabetically	390

Part 3. Appendixes 395

Appendix. Notices	397
Trademarks	398
Bibliography	399

Sending your comments to IBM . . . 401

About this book

This book is a programming reference for the MQSeries Everyplace for Multiplatforms product (generally referred to in this book as MQSeries Everyplace). It contains details of the parameters and calling sequences of the various methods within the MQSeries Everyplace class libraries. This book is intended to be used in conjunction with the MQSeries Everyplace Programming Guide and existing books or manuals on the programming languages that are used to write MQSeries Everyplace programs.

This document is continually being updated with new and improved information. For the latest edition, please see the MQSeries family library Web page at <http://www.ibm.com/software/ts/mqseries/library/>.

Who should read this book

This book is intended for programmers wanting to write MQSeries Everyplace programs for use in a pervasive computing environment.

Prerequisite knowledge

It is assumed that the reader has a working knowledge of the basic programming techniques for the language in which the MQSeries Everyplace programs are to be written

An initial understanding of the concepts of secure messaging is an advantage. If you do not have this understanding, you may find it useful to read the following MQSeries books:

- *MQSeries An Introduction to Messaging and Queuing*
- *MQSeries for Windows NT[®] V5R1 Quick Beginnings*

These books are available in softcopy form from Book section of the online MQSeries library. This can be reached from the MQSeries Web site, URL address <http://www.ibm.com/software/ts/MQSeries/library/>

Terms used in this book

The following terms are used throughout this book:

MQSeries

refers to the following three MQSeries Messaging product groups:

- Distributed messaging
- Host messaging
- Workstation messaging

MQSeries Everyplace

refers to the fourth MQSeries Messaging product group, pervasive messaging.

Device

A computer of any size that is running MQSeries Everyplace programs but *does not have a channel manager* object installed.

Note: For licensing purposes *device* is synonymous with *MQSeries Everyplace client*.

Gateway

A computer of any size that is running MQSeries Everyplace programs and has a *channel manager* object installed.

Note: For licensing purposes *gateway* is synonymous with *MQSeries Everyplace server*.

Summary of Changes

This section describes changes to this edition of *MQSeries Everywhere for Multiplatforms Programming Reference*. Changes since the previous edition of the book are marked by vertical lines to the left of the changes.

Changes for this edition (SC34-5846-01)

In addition to corrections and clarifications, the following information has been added:

- Various adapter classes
- `filterMessage` method added to `QueueRule` class
- List of Exceptions

Part 1. MQSeries Everyplace Java API

Chapter 1. MQSeries Everyplace classes and interfaces

The following chapters contain detailed information about the classes and interfaces supplied with MQSeries Everyplace. The classes are arranged in alphabetical order within the package in which they are delivered.

MQSeries Everyplace contains the following packages:

Note: Classes marked ** are available only in the high security version of MQSeries Everyplace Version 1.1.

com.ibm.mqe

Table 1. Classes in package com.ibm.mqe

Class name	Purpose
MQe	Used to derive other MQSeries Everyplace classes
MQeAdapter	This is the definition for the methods that all MQe adapters must provide. Any new adapters must inherit from MQeAdapter
MQeAdminMsg	Provides a base for administration messages
MQeAttribute	Contains the mechanisms to perform authentication, encryption and compression
MQeChannelListener	Used to create a listener for incoming MQSeries Everyplace logical channels
MQeChannelManager	Creates a manager for the MQSeries Everyplace logical channels
MQeEnumeration	Holds a collection of MQSeries Everyplace message objects
MQeException	Creates an MQeException object
MQeFields	Holds data items and provides mechanisms to dump and restore the data
MQeKey	Creates an MQeKey object that can be attached to and used by an attribute object
MQeMessageEvent	Creates an MQeMessageEvent object that is passed to an application when an MQeMessage event occurs
MQeMsgObject	Holds the data, or contains the necessary logic to obtain the data, to send from one MQSeries Everyplace system to another
MQeQueue	Creates an MQSeries Everyplace queue object
MQeQueueManager	Creates an MQSeries Everyplace queue manager object
MQeQueueManagerConfigure	Used to create and delete queue managers and the default queues.
MQeQueueManagerRules	Contains methods that are invoked when the queue manager performs certain operations

MQSeries Everyplace classes

Table 1. Classes in package *com.ibm.mqe* (continued)

Class name	Purpose
MQeQueueRule	Contains methods that are invoked when certain events occur on queues.

Table 2. Interfaces in package *com.ibm.mqe*

Interface name	Purpose
MQeEventLogInterface	All MQSeries Everyplace log handlers must implement this interface
MQeMessageListenerInterface	This interface must be implemented by all objects that wish to receive MQeMessage events
MQeRunListInterface	An interface that allows a list of MQSeries Everyplace applications to be passed to a queue manager when it is activated
MQeSecurityInterface	An optional interface that allows a Java [®] security manager to authorize or reject a call
MQeTraceInterface	All MQSeries Everyplace trace handlers must implement this interface

com.ibm.mqe.administration

Table 3. Classes in package *com.ibm.mqe.administration*

Class name	Purpose
MQeAdminQueueAdminMsg	Used to manage queues of type MQeAdminQueue
MQeConnectionAdminMsg	Class used to manage connections of type MQeConnectionDefinition
MQeHomeServerQueueAdminMsg	Used to manage queues of type MQeHomeServerQueue
MQeQueueAdminMsg	Used to manage MQSeries Everyplace local queues of type MQeQueue
MQeQueueManagerAdminMsg	Used to manage queue managers of type MQeQueueManager
MQeRemoteQueueAdminMsg	Used to manage remote queues of type MQeRemoteQueue
MQeStoreAndForwardQueueAdminMsg	Used to manage queues of type MQeStoreAndForwardQueue

com.ibm.mqe.attributes

Table 4. Classes in package *com.ibm.mqe.attributes*

Class name	Purpose
**MQe3DESCryptor	Provides mechanisms for 3DES encryption
MQeDESCryptor	Provides mechanisms for DES encryption
MQeGenDH	Creates an MQeDHk.java file from which solution unique MQeDHk class objects can be created

MQSeries Everyplace classes

Table 4. Classes in package *com.ibm.mqe.attributes* (continued)

Class name	Purpose
MQeLocalSecure	Provides a simple local security service
MQeLZWCompressor	Provides mechanisms for LZW compression
**MQeMARSCryptor	Provides mechanisms for MARS encryption
MQeMAttribute	Provide simple message-level protection
**MQeMTrustAttribute	Provides more advanced message-level protection
**MQeRC4Cryptor	Provides mechanisms for RC4 encryption
**MQeRC6Cryptor	Provides mechanisms for RC6 encryption
MQeRleCompressor	Provides mechanisms for Run Length encoded compression
**MQeWTLSertAuthenticator	Provides mechanisms for mini-certificate authentication
MQeXORCryptor	Provides mechanisms for XOR encryption

com.ibm.mqe.registry

Table 5. Classes in package *com.ibm.mqe.registry*

Class name	Purpose
MQePrivateRegistry	Creates a private registry object that provides controlled access to a set of private and public objects
MQePrivateRegistryConfigure	Used to configure a private registry
MQePublicRegistry	Creates a public registry object that provides controlled access to a set of public objects

com.ibm.mqe.server

Table 6. Interfaces in package *com.ibm.mqe.server*

Interface name	Purpose
**MQeMiniCertIssuanceInterface	Used to define the way in which instances of MQeMiniCertificateServerGUI manages new Mini-Certificate issuance

com.ibm.mqe.mqemqmessage

Table 7. Interfaces in package *com.ibm.mqe.mqemqmessage*

Interface name	Purpose
MQeMQMsgObject	Used to represent an MQSeries style message object within MQSeries Everyplace

com.ibm.mqe.mqbridgeTable 8. Classes in package *com.ibm.mqe.mqbridge*

Class name	Purpose
MQeCharacteristicLabels	Groups together all the <i>labels</i> used in any MQeFields object used in the bridge code
MQeClientConnectionAdminMsg	Used to encapsulate an administration command that acts on the MQeClientConnection object.
MQeListenerAdminMsg	Used to encapsulate an administration command that acts on the MQeListener object.
MQeMQBridgeAdminMsg	Used to encapsulate an administration command that acts on the MQeBridge object.
MQeMQBridgeQueue	This queue is used as the interface to the MQS bridge
MQeMQBridgeQueueAdminMsg	Used to administer an MQBridge queue
MQeMQBridges	Loads and maintains all bridge objects associated with a given MQe server"
MQeMQBridgesAdminMsg	Used to encapsulate an administration command that acts on the MQeBridges object.
MQeMQMgrProxyAdminMsg	Used to encapsulate an administration command that acts on the MQeQMGrProxy object.
MQeRunState	Holds the <i>run state</i> of an administered object

Table 9. Interfaces in package *com.ibm.mqe.mqbridge*

Interface name	Purpose
MQeTransformerInterface	All classes that can transform MQMessages to MQeMsgObjects (and vice versa) must conform to this interface

com.ibm.mqe.adaptersTable 10. Classes in *com.ibm.mqe.adapters*

Class name	Purpose
MQeDiskFieldsAdapter	Provides support for reading and writing MQeFields information to a local disk.
MQeMemoryFieldsAdapter	Provides a non-persistent store for MQeFields information
MQeTcpipAdapter	Provides support for reading and writing data over TCP/IP streams.
MQeTcpipHttpAdapter	Extends the MQeTcpipAdapter to provide basic support for the HTTP 1.0 protocol.
MQeTcpipLengthAdapter	Extends the MQeTcpipAdapter to provide a simple, byte efficient protocol.

Table 10. Classes in com.ibm.mqe.adapters (continued)

Class name	Purpose
MQeTcpiHistoryAdapter	Extends the MQeTcpiLengthAdapter to provide a more efficient protocol that caches recently used data.

MQSeries Everyplace classes

Chapter 2. Classes in com.ibm.mqe

This section contains detailed information about the following MQSeries Everyplace classes and interfaces

Table 11. Classes in package com.ibm.mqe

Class name	Purpose
MQe	Used to derive other MQSeries Everyplace classes
MQeAdapter	This is the definition for the methods that all MQe adapters must provide. Any new adapters must inherit from MQeAdapter
MQeAdminMsg	Provides a base for administration messages
MQeAttribute	Contains the mechanisms to perform authentication, encryption and compression
MQeChannelListener	Used to create a listener for incoming MQSeries Everyplace logical channels
MQeChannelManager	Creates a manager for the MQSeries Everyplace logical channels
MQeEnumeration	Holds a collection of MQSeries Everyplace message objects
MQeException	Creates an MQeException object
MQeFields	Holds data items and provides mechanisms to dump and restore the data
MQeKey	Creates an MQeKey object that can be attached to and used by an attribute object
MQeMessageEvent	Creates an MQeMessageEvent object that is passed to an application when an MQeMessage event occurs
MQeMsgObject	Holds the data, or contains the necessary logic to obtain the data, to send from one MQSeries Everyplace system to another
MQeQueue	Creates an MQSeries Everyplace queue object
MQeQueueManager	Creates an MQSeries Everyplace queue manager object
MQeQueueManagerConfigure	Used to create and delete queue managers and the default queues.
MQeQueueManagerRule	Contains methods that are invoked when the queue manager performs certain operations
MQeQueueRule	Contains methods that are invoked when certain events occur on queues.

Table 12. Interfaces in package com.ibm.mqe

Interface name	Purpose
MQeEventLogInterface	All MQSeries Everyplace log handlers must implement this interface
MQeMessageListenerInterface	This interface must be implemented by all objects that wish to receive MQeMessage events

Classes in com.ibm.mqe

Table 12. Interfaces in package com.ibm.mqe (continued)

Interface name	Purpose
MQeRunListInterface	An interface that allows a list of MQSeries Everyplace applications to be passed to a queue manager when it is activated
MQeSecurityInterface	An optional interface that allows a Java security manager to authorize or reject a call
MQeTraceInterface	All MQSeries Everyplace trace handlers must implement this interface

MQe

This class is used to derive other MQSeries Everyplace classes. It contains various useful constant definitions and utility methods to assist with the programming of MQSeries Everyplace . Under normal circumstances applications classes should inherit from this class, for example 'class xxxxx extends MQe'.

Package **com.ibm.mqe**

This class is a descendant of **Object** and implements **Serializable**

Constants

This class provides the following constants:

MQe MsgObject field names

```
public final static String  Msg_CorrelID
public final static String  Msg_MsgID
public final static String  Msg_OriginQMgr
public final static String  Msg_Priority
public final static String  Msg_Time
public final static String  Msg_ReplyToQ
public final static String  Msg_ReplyToQMgr
public final static String  Msg_Style
public final static String  Msg_LockID
public final static String  Msg_Resend
public final static String  Msg_ExpireTime
public final static String  Msg_WrapMsg
```

Message style modifiers

```
public final static int     Msg_Style_Datagram
public final static int     Msg_Style_Request
public final static int     Msg_Style_Reply
```

Standard queue names

```
public final static String  Admin_Queue_Name
public final static String  Admin_Reply_Queue_Name
public final static String  DeadLetter_Queue_Name
public final static String  System_Default_Queue_Name
```

Options for use with MQeAdapter objects

```
public final static String  MQe_Adapter_APPEND
public final static String  MQe_Adapter_BINARY
public final static String  MQe_Adapter_CONTENT
public final static String  MQe_Adapter_FINAL
public final static String  MQe_Adapter_FLUSH
public final static String  MQe_Adapter_HEADER
public final static String  MQe_Adapter_HEADERRSP
public final static String  MQe_Adapter_LENGTH
public final static String  MQe_Adapter_LISTEN
public final static String  MQe_Adapter_PERSIST
public final static String  MQe_Adapter_READ
public final static String  MQe_Adapter_RESET
public final static String  MQe_Adapter_SYNC
public final static String  MQe_Adapter_UNICODE
public final static String  MQe_Adapter_UPDATE
public final static String  MQe_Adapter_WRITE
```

Control Options for use with MQeAdapter objects

```
public final static String  MQe_Adapter_ACCEPT
public final static String  MQe_Adapter_FILENAME
public final static String  MQe_Adapter_FILTER
public final static String  MQe_Adapter_GETPERSIST
```

MQe

```
public final static String MQe_Adapter_LIST
public final static String MQe_Adapter_PULSE
public final static String MQe_Adapter_QOSINPUTS
public final static String MQe_Adapter_SECTION
public final static String MQe_Adapter_SETSOCKET
```

Status Options for use with MQeAdapter objects

```
public final static String MQe_Adapter_BYTECOUNTS
public final static String MQe_Adapter_LOCALHOST
public final static String MQe_Adapter_LINKPARAM
public final static String MQe_Adapter_NETWORK
```

Quality of service field names

```
public final static String QoS_BytesRead
public final static String QoS_BytesWritten
public final static String QoS_Cost
public final static String QoS_DialRetry
public final static String QoS_DialRetryWait
public final static String QoS_Duration
public final static String QoS_ErrorRate
public final static String QoS_Errors
public final static String QoS_Jitter
public final static String QoS_Latency
public final static String QoS_Pulse
public final static String QoS_Rate
public final static String QoS_Retry
public final static String QoS_Size
public final static String QoS_TimeOut
```

Log interface log types

```
public final static byte MQe_Log_SUCCESS
public final static byte MQe_Log_ERROR
public final static byte MQe_Log_WARNING
public final static byte MQe_Log_INFORMATION
```

Exception index numbers

```
public final static int Except_UnCoded
public final static int Except_Debug
public final static int Except_NotSupported
public final static int Except_Syntax
public final static int Except_Type
public final static int Except_Command
public final static int Except_NotFound
public final static int Except_Data
public final static int Except_BadRequest
public final static int Except_Stopped
public final static int Except_Closed
public final static int Except_Duplicate
public final static int Except_NotAllowed
public final static int Except_Rule
public final static int Except_TimeOut
public final static int Except_InvalidHandle
public final static int Except_AllocationFail
public final static int Except_Chnl_Attributes
public final static int Except_Chnl_Destination
public final static int Except_Chnl_Limit
public final static int Except_Chnl_ID
public final static int Except_Chnl_Overrun
public final static int Except_Trnsport_QMgr
public final static int Except_Trnsport_Request
public final static int Except_QMgr_NotActive
public final static int Except_QMgr_InvalidQMGrName
public final static int Except_QMgr_Activated
```

```

public final static int Except_QMgr_AlreadyExists
public final static int Except_QMgr_InvalidQName
public final static int Except_QMgr_QExists
public final static int Except_QMgr_UnknownQMGr
public final static int Except_QMgr_QNotEmpty
public final static int Except_QMgr_QDoesNotExist
public final static int Except_QMgr_QInUse
public final static int Except_QMgr_WrongQType
public final static int Except_QMgr_InvalidChannel
public final static int Except_QMgr_SecureMsgDecodeFailed
public final static int Except_QMgr_NotConfigured
public final static int Except_QMgr_Busy
public final static int Except_Q_NoMsgAvailble
public final static int Except_Q_NoMatchingMsg
public final static int Except_Q_InvalidPriority
public final static int Except_Q_Full
public final static int Except_Q_MsgTooLarge
public final static int Except_Q_NotActive
public final static int Except_Q_Active
public final static int Except_Q_InvalidName
public final static int Except_Q_TargetRegistryRequired
public final static int Except_Uncontactable_DontTransmit
public final static int Except_RasDialFailed
public final static int Except_RasGetProjectionInfoFailed
public final static int Except_RasHangUpFailed
public final static int Except_Connect_AdapterNotActive
public final static int Except_Connect_InvalidDefinition
public final static int Except_Con_AlreadyExists
public final static int Except_Con_AliasAlreadyExists
public final static int Except_Con_AdapterRequired
public final static int Except_Con_InvalidName
public final static int Except_Client_Con_Not_Available
public final static int Except_Reg_NullName
public final static int Except_Reg_AlreadyExists
public final static int Except_Reg_DoesNotExist
public final static int Except_Reg_OpenFailed
public final static int Except_Reg_InvalidSession
public final static int Except_Reg_NotDefined
public final static int Except_Reg_AddFailed
public final static int Except_Reg_DeleteFailed
public final static int Except_Reg_ReadFailed
public final static int Except_Reg_UpdateFailed
public final static int Except_Reg_ListFailed
public final static int Except_Reg_SearchFailed
public final static int Except_Reg_RenameFailed
public final static int Except_Reg_ResetPINFailed
public final static int Except_Reg_CRTKeyDecFailed
public final static int Except_Reg_CRTKeySignFailed
public final static int Except_Reg_DeleteRegistryFailed
public final static int Except_Reg_AlreadyOpen
public final static int Except_Reg_NotSecure
public final static int Except_PrivateReg_BadPIN
public final static int Except_PrivateReg_ActivateFailed
public final static int Except_PrivateReg_NotOpen
public final static int Except_MiniCertReg_BadPIN
public final static int Except_MiniCertReg_ActivateFailed
public final static int Except_MiniCertReg_NotOpen
public final static int Except_PublicReg_ActivateFailed
public final static int Except_PublicReg_InvalidRequest
public final static int Except_Admin_NotAdminMsg
public final static int Except_Admin_ActionNotSupported
public final static int Except_Admin_InvalidField

```

MQe

```
public final static int    Except_Authenticate
public final static int    Except_S_Cipher
public final static int    Except_S_InvalidSignature
public final static int    Except_S_CertificateExpired
public final static int    Except_S_InvalidAttribute
public final static int    Except_S_MiniCertNotAvailable
public final static int    Except_S_RegistryNotAvailable
public final static int    Except_S_BadIntegrity
public final static int    Except_S_NoPresetKeyAvailable
public final static int    Except_S_MissingSection
```

Log record types

```
public final static byte   MQe_Log_Success
public final static byte   MQe_Log_Error
public final static byte   MQe_Log_Warning
public final static byte   MQe_Log_Information
public final static byte   MQe_Log_Audit_Success
public final static byte   MQe_Log_Audit_Failure
```

Events

```
public final static int    Event_Activate
public final static int    Event_Close
public final static int    Event_Logon
public final static int    Event_Logoff
public final static int    Event_QueueManager
public final static int    Event_Queue
public final static int    Event_Attribute
public final static int    Event_Authenticate
public final static int    Event_MiniCert_Validate
public final static int    Event_UserBase
```

Publically accessible variables

debugCall:

When set to **true** causes a stack trace and the contents of the MQeFields object to be written to System.err.println.

```
public static boolean debugCall = false;
```

debugExcept:

When set to **true** a stack trace is printed on System.err.println when certain Exceptions occur within the MQSeries Everyplace system (these exceptions should be handled by a try ... catch ... and would not normally be seen).

```
public static boolean debugExcept = false;
```

debugMQeExcept:

When set to **true**, causes a stack trace to be printed on System.err.println whenever an MQeException is raised.

```
public static boolean debugMQeExcept = false;
```

loader:

This is an object reference to a class loader that enables class files to be dynamically loaded from either the local system or from a remote system.

```
public static MQeLoader loader
```

MQeObjectCount:

This is an integer that contains the current number of instantiated objects that are descendents of the MQe class.

```
public static int MQeObjectCount
```

Method summary

Static methods

Method	Purpose
<code>abbreviate</code>	Returns either the full name or an abbreviated name for the supplied class name.
<code>alias</code>	Adds an alias name for a class name.
<code>asciiToByte</code>	Converts an ASCII string into a byte array
<code>byteToAscii</code>	Converts a byte array into an ASCII string
<code>byteToHex</code>	Converts a byte array to ASCII hex
<code>byteToInt</code>	Converts 4 bytes into an int value
<code>byteToLong</code>	Converts 8 bytes into a long value
<code>byteToShort</code>	Converts 2 bytes into a short value
<code>byteToUnicode</code>	Converts a byte array to a Unicode String
<code>debug</code>	Prints the current call stack to <code>System.err.println</code> . This is used for debugging purposes.
<code>getEventLogHandler</code>	Returns a reference to the currently active log handler or <i>null</i> .
<code>getTraceHandler</code>	Returns a reference to the currently active trace handler or <i>null</i> .
<code>hexToByte</code>	Converts ASCII hex to a byte array
<code>intToByte</code>	Converts an int value into 4 bytes
<code>log</code>	Loads data with the log handler.
<code>mapFileDescriptor</code>	Maps a String to a file descriptor.
<code>setEventLogHandler</code>	Sets the class that will handle log requests.
<code>SetTraceHandler</code>	Sets the class that will process trace messages.
<code>sliceByteArray</code>	Copies a slice out of a byte array
<code>unicodeToByte</code>	Converts a Unicode String to a byte array
<code>unicodeToUTF</code>	Converts a Unicode String to a UTF encoded byte array.
<code>uniqueValue</code>	Generates a unique long value for the current environment.
<code>utfToUnicode</code>	Converts UTF encoded byte array to a Unicode String

Non-Static methods

Method	Purpose
<code>trace</code>	Writes a trace message to either <code>System.out.println</code> or <code>System.err.println</code>
<code>type</code>	Returns a string representation of the class name

MQe abbreviate

Syntax

```
public static String abbreviate(String className, int index )
```

Description

This method resolves an abbreviated class name, or abbreviates a class name.

MQe

An abbreviated class name is of the form "nn:aaaaa" where nn is an number and aaaaa is a string appended to the abbreviation to create a fully qualified class name.

For example, 5:RleCompressor would resolve to com.ibm.mqe.attributes.MQeRleCompressor.

Parameters

ClassName	A String containing the class name or an abbreviated class name
index	An integer. Current supported values are: 0 turn an abbreviated name into a fully qualified class name 1 turn a fully qualified name into an abbreviation

Return values

A String that is either a fully qualified class name or an abbreviated name

Exceptions

java.lang.ArrayOutOfBoundsException You supplied an invalid index

Example

```
class MyApplication
{
    ...
    ...
    String abbrev = MQe.abbreviate( "com.ibm.MQe.Adapters.MQeTcpipHttpAdapater", 1 );
    ...
}
```

MQe alias

Syntax

```
public static void alias( String from, String to )
```

Description

This method assigns or removes an alias name for a class. The **from** parameter is the alias and the **to** parameter is the full class name. To remove an alias set the **to** parameter to *null*.

Parameters

from	A String containing the alias name
to	A String containing the full class name for this alias, or <i>null</i> to remove the alias

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
    ...
    MQe.alias( "Network", "com.ibm.MQe.Adapters.MQeTcpipHttpAdapater" );
    ...
}
```


MQe asciiToByte

Syntax

```
public static byte[] asciiToByte( String data )
```

Description

This method converts a String into a byte array preserving only the low byte of each character.

Parameters

data A String containing the ASCII data

Return values

A byte array containing the ASCII data

Exceptions

none

Example

```
class MyApplication
{
    ...
    ...
    byte data[] = MQe.asciiToByte( "This is some test data" );
    ...
}
```

MQe byteToAscii

Syntax

```
public static String byteToAscii( byte data[] )
```

Description

This method converts a byte array into an ASCII string by copying the byte into the low byte of each character in the String.

Parameters

data A byte array containing the data to be converted

Return values

A String containing the converted data

Exceptions

none

Example

```
class MyApplication
{
    ...
    ...
    String data = MQe.byteToAscii( new byte[] { 64, 65, 66, 67, 68 } );
    ...
}
```

MQe byteToHex

Syntax

```
public static String byteToHex( byte data )
```

```
public static String byteToHex( byte data[], int offset, int count )
```

MQe

Description

This method converts a byte array to a String containing the character Hex representation of the data.

Parameters

data A byte array containing the data to be converted
offset The starting element index of within the **data** array
count The number of elements to be converted

Return values

The HEX String.

Exceptions

none

Example

```
...  
String hexData = byteToHex( ByteArray );  
...
```

MQe byteToInt

Syntax

```
public static int byteToInt( byte data[], int offset )
```

Description

This method converts a byte array to an integer value

Parameters

data A byte array containing the data to be converted
offset The starting element index of within the **data** array

Return values

The HEX String.

Exceptions

none

Example

```
...  
int value = byteToInt( ByteArray );  
...
```

MQe byteToLong

Syntax

```
public static int byteToLong( byte data[], int offset )
```

Description

This method converts a byte array to an integer value

Parameters

data A byte array containing the data to be converted
offset The starting element index of within the **data** array

Return values

The long integer value

Exceptions

none

Example

```
...
    long value = byteToLong( byteArray, 0 );
...
```

MQe byteToShort**Syntax**

```
public static int byteToShort( byte data[], int offset )
```

Description

This method converts a byte array to a short integer value

Parameters

data A byte array containing the data to be converted
offset The starting element index of within the **data** array

Return values

The shortinteger value

Exceptions

none

Example

```
...
    short value = byteToShort( byteArray, 0 );
...
```

MQe byteToUnicode**Syntax**

```
public static String byteToUnicode( byte data[] )
```

Description

This method converts a byte array to a Unicode string

Parameters

data A byte array containing the data to be converted

Return values

The Unicode string

Exceptions

none

Example

```
...
    String data = byteToUnicode( ByteArray );
...
```

MQe debug**Syntax**

```
public static void debug( String data )
```

Description

Causes a stack trace to be written to `System.err.println` followed by the `String data`. Processing continues normally.

MQe

Parameters

data A String containing data to identify this stack print

Return values

none

Exceptions

none

Example

```
class MySampleClass extends MQe
{
    MySampleClass ( )
    {
        ...
        MQe.debug( "" );
        ...
    }
    ...
}
```

MQe getEventLogHandler

Syntax

```
Public static MQeEventLogInterface getEventLogHandler( )
```

Description

Returns the current event log handler object

Parameters

none

Return values

The log handler object or *null*

Exceptions

none

Example

```
class MySampleClass extends MQe
{
    MySampleClass ( )
    {
        ...
        MQeEventLogInterface Logger= MQe.getEventLogHandler( );
        ...
    }
    ...
}
```

MQe getTraceHandler

Syntax

```
Public static MQeTraceInterface getTraceHandler( )
```

Description

Returns the current trace handler object

Parameters

none

Return values

The trace handler object or *null*

Exceptions

none

Example

```

class MySampleClass extends MQe
{
    MySampleClass ( )
    {
        ...
        MQeTraceInterface Logger= MQe.getTraceHandler( );
        ...
    }
    ...
}

```

MQe hexToAscii**Syntax**

```
public static String hexToAscii( String data ) throws Exception
```

Description

This method converts a String containing the character Hex representation of the data to a byte array.

Parameters

data A String containing the data to be converted

Return values

A String array containing the converted data.

Exceptions

none

Example

```

...
String data = hexToAscii( "30313233343536373839" );
...

```

MQe hexToByte**Syntax**

```
public static byte[] hexToByte( String data ) throws Exception
```

Description

This method converts a String containing the character Hex representation of the data to a byte array..

Parameters

data A String containing the data to be converted

Return values

A Byte array containing the converted data.

Exceptions

none

Example

```

...
byte data[] = hexToByte( "30313233343536373839" );
...

```

MQe intToByte

Syntax

```
public static byte[] intToByte( int data )
```

Description

Convert an integer value to 4 bytes of a byte array

Parameters

data An integer containing the data to be converted

Return values

A byte array containing the converted data

Exceptions

none

Example

```
...
byte data[] = intToByte( "30313233343536373839" );
...
```

MQe log

Syntax

```
public static void log( byte logType, int logNumber, Object logData)
```

Description

Sends a message to the event log routine

Parameters

logType a byte containing the type of the log message. For example:

- MQe.MQe_Log_Success
- MQe.MQe_Log_Error
- MQe.MQe_Log_Warning
- MQe.MQe_Log_Information
- MQe.MQe_Log_Audit_Success
- MQe.MQe_Log_Audit_Failure

logNumber an integer identifying the message

logData a String containing the message data to be logged

Return values

none

Exceptions

none

Example

```
...
try
{
    setLogHandler( new MyLogHandler( ... );
    log( MQe.MQe_LogSuccess, 123, "TEST opened" );
    ...
}
catch ( Exception e )
```

```

    {
    log( MQe.MQe_LogError, 123, "TEST failed" );
    }
    ...

```

MQe mapFileDescriptor

Syntax

```

public static void mapFileDescriptor( String filedDesc,
                                     Object newDesc[] )

```

Description

Assigns an alias or nickname to a file descriptor, parameters and options. This method is normally used internally.

Parameters

fileDesc	A String containing a file descriptor
newDesc	An object array containing the new filedescriptor and any parameter and option data

Return values

none

Exceptions

none

Example

```

...
MQe.MapFileDescriptor( "QMgrName", new String[] {
    "TcpipHttp:127.0.0.1:8080",
    "?Channel",
    "" } );
...
...

```

MQe setEventLogHandler

Syntax

```

public static MQeEventLogInterface setEventLogHandler(
    MQeLogInterface logObj )

```

Description

Returns the current event log handler object and sets the new handler so that MQe will use it. The log handler object gets control on all Log requests

Parameters

logObj	A log handler object that conforms to the MQeEventLogInterface
---------------	---

Return values

The previous log handler object or *null*

Exceptions

none

Example

```

class MySampleClass extends MQe
{
    MySampleClass ( )
    {
        super( );
    }
}

```

MQe

```
        setEventLogHandler( new Examples.Log.LogToDiskFile( "ThisFile.log" ) );
        ...
    }
    ...
}
```

MQe setTraceHandler

Syntax

```
public static MQeTraceInterface setTraceHandler( MQeTraceInterface traceObj )
```

Description

Returns the current trace handler object and sets the new handler so that MQe will use it. The trace handle object gets control on all MQe.Trace method calls.

Parameters

traceObj A trace handler object that conforms to the **MQeTraceInterface**

Return values

The previous trace handler object or *null*

Exceptions

none

Example

```
class MySampleClass extends MQe
{
    MySampleClass ( )
    {
        super( );
        setTraceHandler( new MQeTraceWindow( "Window Title", null ) );
        ...
    }
    ...
}
```

MQe sliceByteArray

Syntax

```
public static byte[] sliceByteArray( byte data[],
int offset,
int length )
```

Description

This method returns an array of bytes that consists of the data starting at data[Offset] and has length number of elements.

Note: This is a copy of a portion of the **data** array

Parameters

data Source byte array
offset Starting element within Data to be copied
length Number of bytes to be copied

Return values

A byte array containing a copy of the elements from data.

Exceptions

none

Example

```

class MySampleClass extends MQe
{
    MySampleClass ( )
    {
        ...
        byte data[] = { (byte) 1, (byte) 2, (byte) 3, (byte) 4, (byte) 5, };

        byte temp[] = sliceByteArray( data, 1, 3 );
        ...
    }
    ...
}

```

MQe trace**Syntax**

```

public void trace( String msg )
public void trace(int msgNumber, long insert)
public void trace( int msgNumber, Object insert )

```

Description

Sends a message to the trace routine

Parameters

msg	A String containing a prefix character and the message. The prefix byte consists of: " " user message "I" Information message "W" Warning message "E" Error message "S" Security message "D" Debug message "_ " user message "i" Information message "w" Warning message "e" Error message "s" Security message "d" Debug message
msgNumber	An integer containing the number of the message to be displayed. The message must have previously been added using the MQeTrace.addMessage method. The message number must be in the range $0 \leq \text{msgNumber} \leq 32767$.
insert	Either an integer value or an Object of type String or String[] that is inserted into the point in message template where there is an insert identifier (see the Trace example for details).

Return values

none

Exceptions

MQe

IOException I/O error occurred

Example

```
...
{
  ...
  trace( "I:Information message" );
  trace( 5, "Error message text" );
  ...
}
...
```

MQe type

Syntax

```
public String type( )
```

Description .

Returns the String name of the object.

Note: This may or may not return an abbreviated class name, see the **abbreviate** method.

Parameters

none

Return values

A String containing the name of the object

Exceptions

none

Example

```
...
MQe.MQeMsgObject object = new MQeMsgObject( );
String objectName = object.type();
...
```

MQe unicodeToByte

Syntax

```
public static byte[] unicodeToByte( String data )
```

Description

This method converts a Unicode String to a byte array

Parameters

data A byte array containing the data to be converted

Return values

A Byte array containing the converted data.

Exceptions

none

Example

```
...
byte data[] = unicodeToByte( "This is a Data string" );
...
```

MQe unicodeToUTF

Syntax

```
public static byte[] unicodeToUTF( String data )
```

Description

This method converts a Unicode String to a byte array

Parameters

data A byte array containing the data to be converted

Return values

A byte array containing the converted data.

Exceptions

none

Example

```
...
byte data[] = unicodeToUTF( "This is a Data string" );
...
```

MQe uniqueValue

Syntax

```
public long uniqueValue( )
```

Description .

Returns a long value using the **System.currentTimeMillis()** call, guaranteeing that the value will not have been returned by any previous call to uniqueValue.

Parameters

none

Return values

A long integer value that is a unique number for the current environment.

Exceptions

none

Example

```
...
long number = MQe.uniqueValue( );
...
```

MQe utfToUnicode

Syntax

```
public static String utfToUnicode( byte data[])
```

Description

This method converts a byte array containing UTF encoded Unicode to a Unicode String.

Parameters

data A byte array containing the data to be converted

Return values

The Unicode String

MQe

Exceptions

none

Example

```
...  
String data = MQe.utfToUnicode( byteArray );  
...
```

MQeAdapter

This is the definition for the methods that all MQSeries Everyplace adapters must provide. Any new adapters must inherit from MQeAdapter

Package **com.ibm.mqe**

Method summary

Method	Purpose
activate	Activates the loaded adapter
close	Used to finish with the adapter
control	Performs some adapter specific control function.
checkOption	Used within the file adapters to check for options.
equals	Performs an equality check with the adapter instance.
erase	Erases a file via the adapter.
open	Opens an adapter.
qualityOfService	Returns the qualityOfService object (an MQeFields object).
read	Reads data from the adapter.
readEOF	Reads a complete file, or until an EOF Exception occurs from the adapter.
readln	Reads data up to a new line character from an adapter.
readObject	Reads data from the adapter and return an object.
status	Requests adapter status information.
write	Writes data via an adapter.
writeln	Writes data followed by a new line character to an adapter.
WriteObject	Writes data from an object to an adapter.

MQeAdapter activate

Syntax

```
public void activate( String fileId,
                    Object parameter,
                    Object option,
                    int value1,
                    int value2 ) throws Exception
```

Description

This constructor is used to activate an adapter.

Note: This entry point is meant to be used by the MQSeries Everyplace object library not by application programs.

Parameters

fileId	Identifier of the file
parameter	Any parameters for the adapter, or <i>null</i>
options	Any options for the adapter, or <i>null</i>
value1	An integer value, or -1 to indicate not set
value2	An integer value, or -1 to indicate not set

MQeAdapter

Return values

none

Exceptions

IOException Device not operational or I/O error occurred

MQeAdapter checkOption

Syntax

1. protected boolean checkOption(String what) throws Exception
2. protected boolean checkOption(Object options, String what) throws Exception

Description

This protected method is used when writing new MQSeries Everyplace adapters. The method checks for a matching option and if found will return **true**. There are two forms of the method:

1. Checks the options that were specified on the activate method
2. Checks the options that are supplied in the **options** parameter

Note: This entry point is meant to be used by descendants of MQeAdapter not by application programs.

Parameters

option Options for this operation
what A String containing the option to be checked

Return values

A boolean return code:

true The option was found
false The option was not found

Exceptions

IOException Error closing the file

MQeAdapter close

Syntax

```
public void close( Object options ) throws Exception
```

Description

To unbind from a file.

The MQeAdapter base class throws a "not supported" exception. New adapters should override this method if appropriate.

Note: This entry point is meant to be used by the MQSeries Everyplace object library not by application programs.

Parameters

options Any options for the adapter, or *null*

Return values

none

Exceptions

IOException Device not operational or I/O error occurred

MQeAdapter control

Syntax

```
public Object control(Object options,
                    Object ctrlObj ) throws Exception
```

Description

The MQeAdapter base class throws a "not supported" exception. New adapters should override this method if appropriate.

Note: This entry point is meant to be used by the MQSeries Everyplace object library not by application programs.

Parameters

options Any options for the adapter, or *null*

ctrlObj An object used by the adapter for the control function (specific to each adapter type)

Return values

An object dependent on the adapter type or *null*.

Exceptions

IOException Device not operational or I/O error occurred

MQeAdapter equals

Syntax

```
public boolean equals( Object item )
```

Description

This method is used to perform an equality check with this adapter.

The MQeAdapter base class compares the fileId with the supplied **item** if **item** is a string otherwise the base object equals method is called.

Note: This entry point is meant to be used by the MQSeries Everyplace object library not by application programs.

Parameters

item An Object to be compared against

Return values

A Boolean, either true or false

Exceptions

IOException Device not operational or I/O error occurred

MQeAdapter erase

Syntax

```
public void erase( Object options ) throws Exception
```

Description

This method is used to delete an existing file.

The MQeAdapter base class throws a "not supported" exception. New adapters should override this method if appropriate.

MQeAdapter

Note: This entry point is meant to be used by the MQSeries Everyplace object library not by application programs.

Parameters

options Any options for the adapter, or *null*

Return values

none

Exceptions

IOException Device not operational or I/O error occurred

MQeAdapter open

Syntax

```
public void open( Object options ) throws Exception
```

Description

This method is used to bind to a file via the adapter.

The MQeAdapter base class throws a "not supported" exception. New adapters should override this method if appropriate.

Note: This entry point is meant to be used by the MQSeries Everyplace object library not by application programs.

Parameters

options Any options for the adapter, or *null*

Return values

none

Exceptions

IOException Device not operational or I/O error occurred

MQeAdapter qualityOfService

Syntax

```
public void qualityOfService( Object options ) throws Exception
```

Description

This method is used to get the quality of service object associated with this instance of the adapter.

Note: This entry point is meant to be used by the MQSeries Everyplace object library not by application programs.

Parameters

options Any options for the adapter, or *null*

Return values

A Quality of service object

Exceptions

none

MQeAdapter read

Syntax


```
public byte[] read( Object options,
                  int value0 ) throws Exception
```

Description

This method is used to read a record from the specified file.

The MQeAdapter base class throws a "not supported" exception. New adapters should override this method if appropriate.

Note: This entry point is meant to be used by the MQSeries Everyplace object library not by application programs.

Parameters

options Any options for the adapter, or *null*
value0 The record number to be written or -1

Return values

A QualityOfService byte array containing the data bytes read from the file object

Exceptions

IOException Device not operational or I/O error occurred
EOFException Past end of file for this file

MQeAdapter readEOF**Syntax**

```
public byte[] readEOF( Object options ) throws Exception
```

Description

This method is used to read the file until EOF condition is reached.

The MQeAdapter base class throws a "not supported" exception. New adapters should override this method if appropriate.

Note: This entry point is meant to be used by the MQSeries Everyplace object library not by application programs.

Parameters

options Any options for the adapter, or *null*

Return values

A byte array containing the file data bytes.

Exceptions

IOException Device not operational or I/O error occurred

MQeAdapter readln**Syntax**

```
public String readln( Object options ) throws Exception
```

Description

This method is used to read a record from the specified file.

The MQeAdapter base class throws a "not supported" exception. New adapters should override this method if appropriate.

MQeAdapter

Note: This entry point is meant to be used by the MQSeries Everyplace object library not by application programs.

Parameters

options Any options for the adapter, or *null*

Return values

A String containing the data bytes read from the file.

Exceptions

IOException Device not operational or I/O error occurred

EOFException Past end of file for this file

MQeAdapter readObject

Syntax

```
public Object readObject( Object options ) throws Exception
```

Description

This method is used to read an Object from the specified file.

The MQeAdapter base class throws a "not supported" exception. New adapters should override this method if appropriate.

Note: This entry point is meant to be used by the MQSeries Everyplace object library not by application programs.

Parameters

options Any options for the adapter, or *null*

Return values

An Object containing the data read from the file.

Exceptions

IOException Device not operational or I/O error occurred

EOFException Past end of file for this file

MQeAdapter status

Syntax

```
public Object status( Object options ) throws Exception
```

Description

This method is used to return adapter status information as a String.

The MQeAdapter base class throws a "not supported" exception. New adapters should override this method if appropriate.

Note: This entry point is meant to be used by the MQSeries Everyplace object library not by application programs.

Parameters

options Any options for the adapter, or *null*. All adapters must support as a minimum the following options:

MQe_File_NETWORK

Returns null or the network type (TCPIP for example)

MQe_File_BYTECOUNTS

Returns the number of bytes read and/or written via the adapter

Return values

A String containing the data bytes read from the file.

Exceptions

IOException Device not operational or I/O error occurred

EOFException Past end of file for this file

MQeAdapter write**Syntax**

```
public void write( Object options,
                  int value0,
                  byte data[] ) throws Exception
```

Description

This method is used to write data to the specified file.

The MQeAdapter base class throws a "not supported" exception. New adapters should override this method if appropriate.

Note: This entry point is meant to be used by the MQSeries Everyplace object library not by application programs.

Parameters

options Any options for the adapter, or *null*
value0 The record number to be written or -1
data Byte array containing the data to be written

Return values

none

Exceptions

IOException Device not operational or I/O error occurred

EOFException Past end of file for this file

MQeAdapter writeln**Syntax**

```
public void Writeln( Object options,
                    String data ) throws Exception
```

Description

This method is used to write data to the specified file.

The MQeAdapter base class throws a "not supported" exception. New adapters should override this method if appropriate.

Note: This entry point is meant to be used by the MQSeries Everyplace object library not by application programs.

Parameters

options Any options for the adapter, or *null*

MQeAdapter

data String containing the data to be written.

Return values

none

Exceptions

IOException Device not operational or I/O error occurred

EOFException Past end of file for this file

MQeAdapter writeObject

Syntax

```
public void writeObject( Object options,  
                        Object data ) throws Exception
```

Description

This method is used to write an Object to the specified file.

The MQeAdapter base class throws a "not supported" exception. New adapters should override this method if appropriate.

Note: This entry point is meant to be used by the MQSeries Everyplace object library not by application programs.

Parameters

options Any options for the adapter, or *null*

data An Object containing the data to be written.

Return values

none

Exceptions

IOException Device not operational or I/O error occurred

EOFException Past end of file for this file

MQeAdminMsg

This class is used to create a basic MQeAdminMsg. The class extends MQeMsgObject and provides a base for administration messages. Descendants of this class are created to administer different types of resource.

Package **com.ibm.mqe**

This class is a descendant of **MQeMsgObject**

Constants and variables

MQeAdminMsg provides the following constants and variables in addition to those provided and inherited by MQeMsgObject:

Additional fields in the message

Action:

The administration action to perform (int)
`public final static String Admin_Action;`

Errors: Errors resulting from action (MQeFields)

`public final static String Admin_Errors;`

MaxAttempts:

Maximum number of times request should be tried (int)
`public final static String Admin_MaxAttempts;`

Parms:

Input/Output parameters for/from the action (MQeFields). Contains the characteristics of a managed resource that the action requires or has returned as a result of the action.

`public final static String Admin_Parms;`

RC: Result of action code (byte)

`public final static String Admin_RC;`

Reason:

Reason for failure (unicode)
`public final static String Admin_Reason;`

TargetQMGr:

The name of the target queue manager where the action is to be performed(ascii)

`public final static String Admin_TargetQMGr`

Basic types of administration actions

Create:

Create the resource
`public final static int Admin_Create;`

Delete:

Delete the resource
`public final static int Admin_Delete;`

Inquire:

Return requested characteristics of resource
`public final static int Admin_Inquire;`

InquireAll:

Return all characteristics of resource

MQeAdminMsg

```
public final static int Admin_InquireAll;
```

Update:

Update characteristics of resource

```
public final static int Admin_Update;
```

Field name of the resource to be managed (ascii)

Name: Managed resource name. This is a characteristic of the managed resource, hence the field must reside within the **Admin_Parms** field.

```
public final static String Admin_Name;
```

Java class of the managed resource (ascii)

Class: Managed resource class. This is a characteristic of the managed resource, hence the field must reside within the **Admin_Parms** field.

```
public final static String Admin_Class;
```

Return codes

Fail: Action failed - see **Reason**

```
public final static int RC_Fail;
```

Mixed:

Action was partially successful - see **Reason**

```
public final static int RC_Mixed;
```

Success:

Action was successful

```
public final static int RC_Success;
```

Constructor summary

Constructor	Purpose
MQeAdminMsg	Creates and initializes a default MQeAdminMsg

Method summary

Method	Purpose
characteristics	Returns an MQeFields object containing the characteristics of the resource
create	Sets up an administration message to run the Admin_Create action
delete	Sets up an administration message to run the Admin_Delete action
duplicate	Creates a new message of the type specified by the replyType parameter
getAction	Returns the administration action that is to be or has been performed
getErrorFields	Returns a reference to the error fields object
getFieldInError	Given a field name return any error that occurred when processing the field.
getInputFields	Returns a reference to the input fields object

Method	Purpose
getName	Gets the name of the managed resource
getOutputFields	Returns a reference to the output fields object
getRC	Returns the return code resulting from the action
getReason	Returns the reason for failure if an error occurred
getTargetQMgr	Returns the queue manager where the request is to be processed
inquire	Sets up the administration message to perform the Action_Inquire action
inquireAll	Sets up the administration message to perform the Action_InquireAll action
setAction	Sets the administration action to perform
setName	Sets the name of the resource that the action is to be performed on
setTargetQMgr	Sets the queue manager where the request is to be processed
update	Sets the administration message up to perform the Action_Update action

MQeAdminMsg

Syntax

```
public MQeAdminMsg() throws Exception
```

Description

The constructor creates and initializes a default MQeAdminMsg

Parameters

none

Return Values

none

Exceptions

java.lang.Exception Various

Example

```
class MyApplication
{
    MQeAdminMsg aMsg = new MQeAdminMsg();
}
```

MQeAdminMsg characteristics

Syntax

```
public MQeFields characteristics() throws Exception
```

Description

Returns an MQeFields object containing the characteristics of the resource. The complete set of field names and types for the resource can be determined from the resulting MQeFields object. (It does not contain the value of each characteristic)

Parameters

none

MQeAdminMsg

Return values

Valid characteristics of the resource

Exceptions

java.lang.Exception Various

Example

```
class MyApplication
{
    MQeFields chars = msg.characteristics();
    Enumeration fields = chars.fields()
    while ( fields.hasMoreElements() )
    {
        System.out.println( "Contains field: "+
            (String)fields.nextElement() );
    }
}
```

MQeAdminMsg create

Syntax

```
public void create( MQeFields parms ) throws Exception
```

Description

Sets up an administration message to run the Admin_Create action. Attempts to create a new managed resource with characteristics as specified in the **parms** parameter.

Parameters

parms An MQeFields object containing name value pairs for any characteristics that require a setting different to the managed resources default setting. The name of the resource can be included in **parms** but can also be set with the **setName** method.

Return values

none

Exceptions

java.lang.Exception Various

Example

```
class MyApplication
{
    ...
    // Create ExampleQ
    MQeFields parms = new MQeFields();
    msg.setName( "ExampleQM", "ExampleQ" );
    parms.putUnicode( MQeQueueAdminMsg.Queue_Description,
        "a new description ..." );

    // Set the action required and its parameters
    // into the message
    msg.create( parms );
}
```

MQeAdminMsg delete

Syntax

```
public void delete( MQeFields parms ) throws Exception
```


Description

Sets up an administration message to run the Admin_Delete action. Attempts to delete a managed resource.

Parameters

parms An MQeFields object which must contain the name of the managed resource to delete if it has not been set with the **setName** method.

Return values

none

Exceptions

java.lang.Exception Various

Example

```
class MyApplication
{
    ...
    // Delete ExampleQ
    MQeFields parms = new MQeFields();
    msg.setName( "ExampleQM", "ExampleQ" );
    msg.delete( parms );
}
```

MQeAdminMsg duplicate

Syntax

```
public MQeFields duplicate( String replyType ) throws Exception
```

Description

Create a new message of the type specified by the replyType parameter. If *null*, a message that is the same type as this message is returned. All fields are duplicated with the exception of fields that constitute the unique message ID.

Note: The **MQeFields.copy** method is employed so only a shallow copy of the message is made.

Parameters

replyType The type of message to be returned or *null* if the same as this message

Return values

Duplicate message

Exceptions

ClassNotFoundException

Example

```
class MyApplication
{
    // Create a message as the same type as this one
    MQeQueueAdminMsg reply =
        (MQeQueueAdminMsg).requestMsg.duplicate( null );
}
```

MQeAdminMsg

MQeAdminMsg getAction

Syntax

```
public int getAction( )
```

Description

Returns the administration action that is to be, or has been performed

Parameters

none

Return values

Field Admin_Action from the MQeAdminMsg or Action_Unknown if not set

Exceptions

none

Example

```
class MyApplication
{
    ...
    int action = requestMsg.getAction();
    switch ( action )
    {
        case Create :
            performCreate();
            break;
        case Delete :
            performDelete();
            ...
    }
}
```

MQeAdminMsg getErrorFields

Syntax

```
public MQeFields getErrorFields()
```

Description

Returns a reference to the error fields object.

Error fields contain any errors related to subproblems that occurred when processing the action. For instance, if a request was made to update 2 characteristics and 1 request succeeds and the other fails, ErrorFields contain the details of the one that failed. The name of the field in error matches that in the Admin_Parms field.

Use the **getRC** method to check the overall result of the action.

Parameters

none

Return values

An empty MQeFields object or the Admin_Errors field from the MQeAdminMsg

Exceptions

none

Example

```
class MyApplication
{
    if ( replyMsg.getRC() != 0 )
```

```

    {
    MQeFields errs = replyMsg.getErrorFields();
    Enumeration fields = errs.fields()
    while ( fields.hasMoreElements() )
    {
    String errF = (String)fields.nextElement()
    System.out.println( "Field: "+
        errF+
        "failed with error "+
        fields.getAscii( Msg_RC ) );
    }
    }
}

```

MQeAdminMsg getFieldInError

Syntax

```
public String[] getFieldInError( String fieldName )
```

Description

This method is used to obtain information on individual errors after a getRC return of RC_Fail or RC_Mixed. Given a field name return any error that occurred when processing the field. If the field that was processed was an array then a corresponding string array is returned that contains the same number of elements. If the field that was processed was not an array then the returned array will only contain one element. If the field was not in error then *null* is returned.

Parameters

fieldname Name of field to test for error

Return values

A string array containing any error that occurred when processing the named field

Exceptions

none

Example

```

class MyApplication
{
    if ( replyMsg.getRC() != 0 )
    {
        String fieldName = MQeQueueAdminMsg.Queue_Priority
        String[] errs = replyMsg.getFieldInError( fieldName );
        if ( errs != null )
            System.out.println( "Error setting priority"+ errs[0]
        )
    }
}

```

MQeAdminMsg getInputFields

Syntax

```
public MQeFields getInputFields()
```

Description

Returns a reference to the input fields object. The input fields object contains the input parameters required by an action.

Parameters

none

MQeAdminMsg

Return values

Reference to an MQeFields object that contains input parameters required for an action

Exceptions

none

Example

```
class MyApplication
{
    MQeFields parms = requestMsg.getInputFields()
}
```

MQeAdminMsg getMaxAttempts

Syntax

```
public int getMaxAttempts( )
```

Description

Get the maximum number of times the request should be retried in the event that the request is held up due to the resource being unavailable at the time the request is processed.

Parameters

none

Return values

Field Admin_MaxAttempts from the MQeAdminMsg is returned or a default of 1 if not set

Exceptions

none

Example

```
class MyApplication
{
    ...
    int tries = requestMsg.getMaxAttempts();
    ...
}
```

MQeAdminMsg getName

Syntax

```
public String getName( )
```

Description

Get the name of the managed resource or *null* if not set

Parameters

none

Return values

Field Admin_Name from the MQeAdminMsg or *null* if not set

Exceptions

none

Example

```

class MyApplication
{
    ...
    String name = requestMsg.getName();
    ...
}

```

MQeAdminMsg getOutputFields

Syntax

```
public MQeFields getOutputFields()
```

Description

Returns a reference to the output fields object. OutputFields contains both the input parameters of a request together with the results of the request.

Parameters

none

Return values

Results of an action

Exceptions

none

Example

```

class MyApplication
{
    MQeFields parms = replyMsg.getOutputFields()
    if (parms.contains( MQeQueueAdminMsg.desc ) )
    {
        System.out.println("Queue description: "+
            parms.getUnicode(MqeQueueAdminMsg.Desc) );
    }
}

```

MQeAdminMsg getRC

Syntax

```
public int getRC( ) throws Exception
```

Description

Returns the code resulting from the action

Parameters

none

Return value

Return code.

Possible values are:

```

public final static int RC_Success;
public final static int RC_Fail;
public final static int RC_Mixed;

```

Exceptions

java.lang.Exception Various

Example

```

class MyApplication
{
    ...
    int rc = ReplyMsg.getRC();
}

```

MQeAdminMsg

```
        if (rc != ReplyMsg.RC_success)
            String error = replyMsg.getReason();
        ....
    }
```

MQeAdminMsg getReason

Syntax

```
public String getReason( )
```

Description

Returns the reason for failure if an error occurred

Parameters

none

Return values

String, typically the exception that caused the failure. If the exception is of type MQeException the string will include the MQeException code at the start of the string as "Code=nnn;"

Exceptions

none

Example

```
class MyApplication
{
    ...
    int rc = replyMsg.getRC();

    if (rc != replyMsg.RC_success)
        String error = replyMsg.getReason();
    ...
}
```

MQeAdminMsg getTargetQMgr

Syntax

```
public String getTargetQMgr( ) throws MQeException
```

Description

Returns the queue manager where the request is to be processed.

Parameters

none

Return values

Queue manager where the request is to be processed.

Exceptions

MQeException	Except_Type, "wrong field type"
	Except_NotFound, Item + " not found".

Example

```
class MyApplication
{
    try
    {
        String targetQMgr = requestMsg.getTargetQMgr();
    }
    catch ( MQeException e)
```

```

    {
        System.out.println("Target queue manager not set")
    }
}

```

MQeAdminMsg inquire

Syntax

```
public void inquire( MQeFields parms ) throws Exception
```

Description

Sets up the administration message to perform the Action_Inquire action

Parameters

parms The names of the characteristics of the managed resource that are to be inquired on. The managed resource name can also be included in the parameters if not set via the **setName()** method.

Return values

none

Exceptions

NullPointerException

Example

```

class MyApplication
{
    ...
    // Request the value of description and max queue depth
    MQeFields parms = new MQeFields();
    parms.putUnicode( MQeQueueAdminMsg.Queue_Description, null );
    parms.putInt( MQeQueueAdminMsg.Queue_MaxQDepth, 0 );

    // set the name of the queue to inquire on
    msg.setName( "ExampleQM", "ExampleQ" );

    // Set the action required and its parameters
    // into the message
    msg.inquire( parms );
}

```

MQeAdminMsg inquireAll

Syntax

```
public void inquireAll( MQeFields parms ) throws Exception
```

Description

Sets up the administration message to perform the Action_InquireAll action. The InquireAll action returns all characteristics of the managed resource.

Parameters

parms needs to contain the name of the resource to be inquired on or *null* if the name has been set via the **setName()** method.

Return values

none

Exceptions

NullPointerException

MQeAdminMsg

Example

```
class MyApplication
{
    ...
    // set the name of the queue to inquire on
    msg.setName( "ExampleQM", "ExampleQ" );

    // Set the action required and its parameters
    // into the message
    msg.inquireAll( new MQeFields() );
}
```

MQeAdminMsg setAction

Syntax

```
public void setAction(int action )
```

Description

Sets the administration action to be performed. Sets the Admin_Action field in the MQeAdminMsg

Parameters

action	Possible values are:
	public final static int Action_Create;
	public final static int Action_Delete;
	public final static int Action_Inquire;
	public final static int Action_InquireAll;
	public final static int Action_Update;
	// additional actions can implemented in subclass

Return values

none

Exceptions

java.lang.Exception	Various
----------------------------	---------

Example

```
class MyApplication
{
    ...
    MQeAdminMsg requestMsg = new MQeAdminMsg()
    requestMsg.setAction(MQeAdminMsg.Action_Inquire);
    ...
}
```

MQeAdminMsg setName

Syntax

```
public void setName( String resourceName ) throws Exception
```

Description

Sets the name of the resource that the action is to be performed on.

Parameters

resourceName Name of the resource

Return values

none

Exceptions

`java.lang.Exception` Various

Example

```
class MyApplication
{
    ...
    // Delete a queue
    MQeFields parms = new MQeFields();

    // Set the action required and its parameters
    // into the message
    MQeQueueManagerAdminMsg msg = new MQeQueueManagerAdminMsg();
    msg.inquireAll( parms );
    msg.setName( "ExampleQM" );
    ...
}
```

MQeAdminMsg setTargetQMgr

Syntax

```
public String setTargetQMgr( String targetQMgr ) throws Exception
```

Description

Sets the queue manager where the request is to be processed.

Parameters

targetQMgr Name of the queue manager that will process the request

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    MQeQueueAdminMsg requestMsg = new MQeQueueAdminMsg();
    requestMsg.setTargetQMgr("ExampleQM");
    requestMsg.setName("ExampleQM", "ExampleQ" );
    requestMsg.create( new MQeFields() );
}
```

MQeAdminMsg update

Syntax

```
public void update( MQeFields parms ) throws Exception
```

Description

Sets up the administration message to perform the Action_Update action, which attempts to update a managed resources characteristics based on those in **parms**.

Parameters

parms The characteristics that are to be updated. If the name of the resource to be managed has not been set then it can be included in parameters.

Return values

none

Exceptions

NullPointerException.

MQeAdminMsg

Example

```
class MyApplication
{
    ...
    // Setname of resource to be managed
    msg.setName( "ExampleQM", "ExampleQ" );

    // Change the value of description
    MQeFields parms = new MQeFields();
    parms.putUnicode( MQeQueueAdminMsg.Queue_Desc, "Change description ... );

    // Set the action required and its parameters
    // into the message
    msg.update( parms );
}
```

MQeAttribute

This class is used to create an attribute object. This object contains the mechanisms to perform authentication, encryption and compression. Attribute objects can be associated with channels, queues, messages, and MQeFields objects.

Package **com.ibm.mqe**

This class is a descendant of **MQe**

Constructor summary

Constructor	Purpose
MQeAttribute	Constructs an MQeAttribute object

Method summary

Method	Purpose
authenticatedID	Returns a String that is the authenticated identifier
activate	Activates an MQeAttribute object
change	Used to change the characteristics of this attribute object
close	Release any resources used by this object
decodeData	Decrypt and/or decompress the supplied data
encodeData	Compress and/or encrypt the supplied data
equals	Compares the settings for one attribute object with this one
getAuthenticator	Gets the object reference to the authenticator
getCompressor	Gets the object reference to the compressor
getCryptor	Gets the object reference to the cryptor.

MQeAttribute

Syntax

1. `public MQeAttribute()`
2. `public MQeAttribute(MQeAuthenticator authenticator,
 MQeCryptor cryptor,
 MQeCompressor compressor
) throws Exception`

Description

Constructs an MQeAttribute object. There are two forms of constructor:

1. Creates an object that requires the attributes to be set with the activate method call
2. Creates a object and automatically calls the activate method

Parameters

- authenticator** An object reference to an MQeAuthenticator object
- cryptor** An object reference to an MQeCryptor object
- compressor** An object reference to an MQeCompressor object

Return values

none

MQeAttribute

Exceptions

MQeException	Various activation errors
IOException	Various IO errors depending on protocol type

Example

```
class MySampleClass
{
    ...
    MQeAttribute attribute = new MQeAttribute( null,
                                              new MQeXorCryport( ),
                                              new MQeRleCompressor( ) );
    ...
    MQeChannel channel = new MQeChannel( aAttribute,
                                         "HTTP://test.server.ibm.com:8080" );
    ...
}
```

MQeAttribute activate

Syntax

```
public void activate( MQeRule rule,
                    MQeAuthenticator authenticator,
                    MQeCryptor cryptor,
                    MQeCompressor compressor) throws Exception
```

Description

Activates an MQeAttribute object.

Parameters

rule	An object reference to an MQeRule object to be used by this attribute.
authenticator	An object reference to an MQeAuthenticator object
cryptor	An object reference to an MQeCryptor object
compressor	An object reference to an MQeCompressor object

Return values

none

Exceptions

MQeException	Various activation errors
IOException	Various IO errors depending on protocol type

Example

```
class MySampleClass
{
    ...
    MQeAttribute attribute = new MQeAttribute( null,
                                              new MQeXorCryport( ),
                                              new MQeRleCompressor( ) );
    ...
    MQeChannel channel = new MQeChannel( attribute,
                                         "HTTP://test.server.ibm.com:8080" );
    ...
}
```

MQeAttribute authenticatedID

Syntax

```
public String authenticatedID( )
```

Description

This method returns a String that is the authenticated id, or *null* if it is not authenticated. This is typically used on the server side of a channel if there is data or a process that is allowed to be run only by certain users.

Parameters

none

Return values

A String that is the authenticated identifier or *null*

Exceptions

none

MQeAttribute change

Syntax

```
public synchronized void change( MQeChannel channel,
                                 MQeRule rule,
                                 MQeAttribute attribute) throws Exception
```

Description

This method is called to change the characteristics of an attribute object. That is, to change the rule, authenticator, cryptor or compressor used by the attribute object. If the Channel parameter is not *null*, the remote end of the channel has to agree to the change of characteristics otherwise an exception is thrown.

Parameters

channel	An object reference to a channel used for any communications
rule	A rules object used to verify that the change is allowed. Note: The previous rules object has to allow the new rules object.
attribute	An object reference to an MQeAttribute

Return values

none

Exceptions

MQeException	Except_Rule, "Disallowed by rule"
---------------------	-----------------------------------

Depends on the authenticator, cryptor and/or the compressor used by the attribute

MQeAttribute close

Syntax

```
public void close( ) throws Exception
```

Description

Closes and releases resources used by the authenticator.

MQeAttribute

Parameters

none

Return values

none

Exceptions

MQeException

Invalid or not allowed

MQeAttribute decodeData

Syntax

```
public byte[] decodeData( MQeChannel channel,  
                          byte data[],  
                          int offset,  
                          int count ) throws Exception
```

Description

This method is called to decode (decrypt and/or decompress) the bytes referenced by **data**, **offset** and for length **count**.

Note: This method is intended for internal use and is not normally called by applications.

Parameters

channel An object reference to the channel used to receive the encoded data or *null*

data An object reference to a byte array containing the data to be decoded

offset An integer index specifying the start byte in the data array

count An integer count of the number of bytes to decode

Return values

none

Exceptions

Depends on the authenticator, cryptor and/or the compressor used by the attribute

MQeAttribute encodeData

Syntax

```
public byte[] encodeData( MQeChannel channel,  
                          byte data[],  
                          int offset,  
                          int count ) throws Exception
```

Description

Is called to encode (encrypt and/or compress) the bytes referenced by **data**, **offset** and, for length, **count**.

Note: This method is intended for internal use and is not normally called by applications.

Parameters

channel An object reference to the channel used to transmit the encoded data or *null*

data	An object reference to a byte array containing the data to be encoded
offset	An integer index specifying the start byte in the data array
count	An integer count of the number of bytes to encode

Return values
none

Exceptions
Depends on the authenticator, cryptor and/or the compressor used by the attribute

MQeAttribute equals

Syntax

```
public boolean equals( Object thisItem )
```

Description

This method is called to compare **thisItem** with this attribute object for equality.

Parameters

thisItem An object reference, normally to an MQeAttribute object

Return values

A boolean value:

true implies they compare equal

false implies they are not equal

Exceptions

Depends on the authenticator, cryptor and/or the compressor used by the attribute

MQeAttribute getAuthenticator

Syntax

```
public MQeAuthenticator getAuthenticator( )
```

Description

Is called to return the object reference to the authenticator used by this attribute, or *null* if there is no authenticator.

Parameters

none

Return values

An MQeAuthenticator object reference or *null*.

Exceptions

none

MQeAttribute getCompressor

Syntax

```
public MQeCompressor getCompressor( )
```

Description

Is called to return the object reference to the compressor used by this attribute, or *null* if there is no compressor.

MQeAttribute

Parameters

none

Return values

An MQeCompressor object reference or *null*.

Exceptions

none

MQeAttribute getCryptor

Syntax

```
public MQeCryptor getCryptor( )
```

Description

Is called to return the object reference to the cryptor used by this attribute, or *null* if there is no cryptor.

Parameters

none

Return values

An MQeCryptor object reference or *null*

Exceptions

none

MQeChannelListener

Licence warning

The use of this class is subject to the following conditions:

- If MQSeries Everyplace is purchased for use as a *device* (client) it may **not** be used to create a *channel listener*.
- The presence of a *channel listener* defines a *gateway* environment which requires a gateway (server) licence.

This class is used to create a listener for incoming MQSeries Everyplace logical channels.

Package **com.ibm.mqe**

This class is a descendant of **MQe**

Constructor summary

Constructor	Purpose
MQeChannelListener	used to create a listener for incoming MQSeries Everyplace logical channels

Method summary

Method	Purpose
activate	Activates the channel listener if it was not activated by the class constructor.
setTimer	Called to set up a time-out interval for any channels accepted by this channel listener.
stop	Called to stop the channel listener accepting any new inbound requests.

MQeChannelListener

Syntax

1. public MQeChannelListener()
2. public MQeChannelListener (Object listener,
String fileType,
Object processor)

Description

Constructs an MQeChannelListener object. This is the class that handles incoming MQeChannel requests if not running under the control of a Server (e.g. WebSphere). There are two forms of the constructor:

1. With no parameters. The class is instantiated but not activated. In order to activate the class the **activate** method must be called.
2. With parameters that define:
 - The Listening adapter , for example @Network::80@

Note: for the TCPIP adapters the @adapter::port_no@ means listen

- The file type to be used when an incoming request is accepted, for example @Network:@

MQeChannelListener

- A class instance that processes the channel requests. Normally this is an instance of **MQeChannelManager**

Parameters

listener	An object defining either an MQeAdapter object or a file descriptor string used to listen for incoming requests
fileType	A String defining the file descriptor to be used in creating a new instance of an MQeAdapter object used to read and write data for the new channel
processor	An instance of an object to be used to manage the channels, normally an instance of MQeChannelManager

Return values

none

Exceptions

none.

Example

```
class MySampleClass
{
    ...
    MQeChannelListener c1 = new MQeChannelManager( @Network::8080@,
                                                @Network:@,
                                                new MQeChannelManager( ) );
    ...
}
```

MQeChannelListener activate

Syntax

```
public void activate( Object listener,
                    String fileType,
                    Object processor )
```

Description

Activates an MQeChannelListener object. This is normally used only if the class was instantiated using a constructor with no parameters. The parameters define:

- The Listening adapter, for example @Network::80@

Note: For the TCPIP adapters the @adapter::port_no@ means listen

- The file type to be used when an incoming request is accepted, for example @Network:@
- A class instance that processes the channel requests. Normally this is an instance of **MQeChannelManager**

Parameters

listener	An object defining either an MQeAdapter object or a file descriptor string used to listen for incoming requests
fileType	A String defining the file descriptor to be used in creating a new instance of an MQeAdapter object used to read and write data for the new channel
processor	A instance of an object to be used to manage the channels, normally an instance of MQeChannelManager

Return values

none

Exceptions

none

Example

```
class MySampleClass
{
    ...
    MQeChannelListener cl = new MQeChannelManager( );
    cl.activate( QNetwork::8080Q, QNetwork:Q, new MQeChannelManager( ) );
    ...
}
```

MQeChannelListener setTimer**Syntax**

```
public void setTimer( int interval ) throws Exception
```

Description

This method is used to set a channel time-out interval for any channels accepted by this channel listener.

Parameters

interval	An integer value in seconds of the desired time out interval
-----------------	--

Return values

none

Exceptions

MQeException	Invalid channel or not allowed
IOException	Error performing I/O operations

Example

```
class MySampleClass extends MQe
{
    MQeChannelListener cl = new MQeChannelManager( "Network::8080",
                                                    "Network:",
                                                    new MQeChannelManager( ) );
    ...
    cl.setTimer( 300 );
    ...
    ...
}
```

MQeChannelListener stop**Syntax**

```
public void stop( )
```

Description

Used to stop the channel listener accepting any new channel requests

Parameters

none

Return values

none

MQeChannelListener

Exceptions

none.

Example

```
class MySampleClass
{
    MQeChannelListener cl = new MQeChannelManager( "Network:8080",
                                                    "Network:",
                                                    new MQeChannelManager( ) );
    ...
    cl.stop( );
    ...
}
```

MQeChannelManager

Licence warning

The use of this class is subject to the following conditions:

- If MQSeries Everyplace is purchased for use as a *device* (client) it may **not** be used to create a *channel manager*.
- The presence of a *channel manager* defines a *gateway* environment which requires a gateway (server) licence.

This class is used to create a manager for the MQSeries Everyplace logical channels.

Package

`com.ibm.mqe`

This class is a descendant of `MQe`.

Constructor summary

Constructor	Purpose
<code>MQeChannelManager</code>	Constructs a <code>MQeChannelManager</code> object.

Method summary

Method	Purpose
<code>getGlobalHashtable</code>	Called to get a reference to the Hashtable used to hold any shared objects.
<code>mapDestination</code>	Called to set up a reroute for one destination to another.
<code>numberOfChannels</code>	Called to get the number of currently active logical channels.
<code>process</code>	Called to process data (bytes) received, destined for an MQSeries Everyplace logical channel.
<code>timeOut</code>	Called to force any logical channels to be timed out if they have been idle for more than a specified interval.
<code>totalNumberOfChannels</code>	Called to get the total number of channels that have been used since the channel manager was activated.

MQeChannelManager

Syntax

```
public MQeChannel( )
```

Description

Constructs a `MQeChannelManager` object.

Parameters

none

Return values

none

Exceptions

none

MQeChannelManager

Examples

```
class MySampleClass
{
    ...
    MQeChannelManager cm = new MQeChannelManager( );
    ...
}
```

MQeChannelManager getGlobalHashtable

Syntax

```
public Hashtable getGlobalHashTable( )
```

Description

Returns the Global hashtable belonging to this instance of the channel manager. This table can be used to hold information that persists across channels

Parameters

none

Return Values

none

Exceptions

none

Example

```
class MySampleClass
{
    try
    {
        MQeChannelManager cm = new MQeChannelManager( );
        Hashtable table = cm.getGlobalHashtable( );
        ...
    }
    catch ( Exception e )
    {
    }
    ...
}
```

MQeChannelManager mapDestination

Syntax

```
public void mapDestination(String destination,
                          String newDestination)
```

Description

This method is used to set up a route from **destination** to **newDestination**.

Parameters

destination A string defining the destination to be remapped

newDestination A string defining the new destination

Return values

none

Exceptions

none

Example

```

class MySampleClass
{
    try
    {
        MQeChannelManager cm = new MQeChannelManager( );
        cm.mapDestination( "One", "Two" );
        ...
    }
    catch ( Exception e )
    {
    }
    ...
}

```

MQeChannelManager numberOfChannels**Syntax**

```
public int numberOfChannels( int newLimit )
```

Description

This method returns the number of currently active channels.

Parameters

newLimit The new maximum number of concurrent channels allowed by this channel manager, a value of 0 implies no limit

Return Values

An integer value that is the number of current channels.

Exceptions

none

Example

```

...
MQeChannelManager cm = new MQeChannelManager( );
int count = cm.numberOfChannels( 0 );
...
...

```

MQeChannelManager process**Syntax**

1. `public void process(MqeAdapter adapter) throws Exception`
2. `public void process(MqeAdapter adapter, byte data[]) throws Exception`

Description

There are two forms of the process method :

1. An MQeAdapter object as the only parameter. This is used to read the data to be passed on to the logical channel.
2. An MQeAdapter (or *null*) and an array of bytes. The array contains the data to be processed by the logical channel.

Parameters

adapter An **MQeAdapter** object to be used for any I/O operations

data A byte array containing the data to be processed

MQeChannelManager

Return values

none

Exceptions

MQeException

Invalid channel or not allowed

data

A byte array containing the data to be processed

Example

```
class MySampleClass extends MQe
{
    try
    {
        MQeChannelManager cm = new MQeChannelManager( );
        ...
        cm.process( null, data );
        ...
    }
    catch ( Exception e )
    {
    }
    ...
}
```

MQeChannelManager timeOut

Syntax

1. public void timeOut(long age)
2. public void timeOut(MQeChannel channel,long age)

Description

This method is used to check all channels or one specific channel to see if they have been idle for more than **age** milliseconds. Any channels that have exceeded this time are closed

Parameters

age An interval in milliseconds. If the channel has been idle for more than this interval it is considered to have timed out, and is closed

channel A specific MQeChannel to be check to see if it has timed out

Return Values

none

Exceptions

none

Example

```
...
cm.timeOut( 30 * 60 * 1000 );
...
```

MQeChannelManager totalNumberOfChannels

Syntax

```
public long totalNumberOfChannels( )
```


Description .

This method returns the total number of channels that have been used since the channel manager was activated.

Parameters

none

Return Values

A long integer value that is the total number of channels.

Exceptions

none

Example

```
MQeChannelManager cm = new MQeChannelManager( );  
long count = cm.totalNumberOfChannels( );  
...  
...
```

MQEnumeration

This class is used to hold a collection of MQSeries Everyplace message objects. It allows the messages to be enumerated in an identical manner to the standard Java Enumeration class.

Package **com.ibm.mqe**

Extends **java.util.Enumeration**.

Method summary

Method	Purpose
getLockId	Returns the lock id associated with this group of messages, if one exists.
getNextMessage	Returns the next message in the enumeration.
getQueueManagerName	Returns the name of the queue manager that owns the queue from which the messages contained in the enumeration were browsed.
getQueueName	Returns the name of the queue from which the messages contained in the enumeration were browsed.

MQEnumeration getLockId

Syntax

```
public long getLockId()
```

Description

If a there is a lock id associated with the group of messages contained in this enumeration, it is returned by this method. The lock id is only set if this enumeration is the result of a **browseMessagesAndLock** operation. Otherwise this method returns a dummy value of "-1".

Parameters

none

Return Values

A long value containing the lock id of the group of messages contained within this enumeration.

Exceptions

none

Example

```
class MyMQApplication
{
    ...
    /* Lock all msgs on this queue */
    MQEnumeration msgEnum = QMgr.browseMessagesAndLock( null, "MyQueue",
                                                         null, null, 0, false );
    long lockId = msgEnum.getLockId(); /* get the Lock Id */
    ...
}
```

MQEnumeration getNextMessage

Syntax

```
public MQEObject getNextMessage( MQAttribute attribute,
                                long confirmId ) throws Exception
```

Description

This method returns the next message in the enumeration. However, the behaviour of this method is dependent upon the justUID parameter of the browse request that created this enumeration. The justUID parameter determines whether the enumeration contains just the unique ID fields of the messages matched by the browse, or all of the fields contained in each message.

If the browse request's justUID parameter is set to true, this method returns the next message in the enumeration (in this case it works identically to the nextElement() method).

If the browse request's justUID parameter is set to false, this method returns the message by issuing a get message command against the target queue. This causes the message to be removed from the target queue.

Use the nextElement() method (inherited from **java.util.Enumeration**) to return a message without removing it from the target queue.

Parameters

attribute	An MQAttribute object used to provide message-level security. The attribute must match the attribute attached to the message returned by this method. Failure to do this may result in message loss.
confirmId	A long value denoting whether or not to use assured message delivery. A nonzero value does not remove the message from the target queue, this occurs on a subsequent confirm flow. A value of zero removes the message from the target queue immediately.

Return Values

An **MQEObject** containing the next element in the enumeration

Exceptions

Except_NotSupported

Example

```
class MyMQApplication
{
    ...
    MQEnumeration msgEnum = null;
    msgEnum = qmgr.browseMessages( "RemoteQMGr", "RemoteQueue", null, null,
                                   false );
    while( msgEnum.hasMoreElements() )
    {
        /* get message */
        MQEObject msg = msgEnum.getNextMessage( null, MQE.uniqueValue() );
        /* confirm get */
        qmgr.confirmGetMessage( msgEnum.getQueueManagerName(),
                               msgEnum.getQueueName(),
                               msg.getMsgUIDFields() );
    }
    ...
}
```

MQEnumeration

MQEnumeration getQueueManagerName

Syntax

```
public String getQueueManagerName()
```

Description

This method returns the name of the queue manager that owns the queue from which the messages contained in the enumeration were browsed.

Parameters

none

Return Values

A String containing the name of the queue manager that owns the queue from which these messages were browsed.

Exceptions

none

Example

```
class MyMQApplication
{
    ...
    MQEnumeration msgEnum = null;
    msgEnum = qmgr.browseMessages( "RemoteQMgr", "RemoteQueue", null, null,
                                  false );
    while( msgEnum.hasMoreElements() )
    {
        /* get message */
        MQMsgObject msg = msgEnum.getNextMessage( null, MQE.uniqueValue() );
        /* confirm get */
        qmgr.confirmGetMessage( msgEnum.getQueueManagerName(),
                                msgEnum.getQueueName(), msg.getMsgUIDFields() ); }
    ...
}
```

Related functions

[getQueueName](#)

MQEnumeration getQueueName

Syntax

```
public String getQueueName()
```

Description

This method returns the name of the queue from which the messages contained in the enumeration were browsed.

Parameters

none

Return Values

A String containing the name of the queue from which these messages were browsed.

Exceptions

none

Example

```
class MyMQApplication
{
    ...
    MQEnumeration msgEnum = null;
    msgEnum = qmgr.browseMessages( "RemoteQMgr", "RemoteQueue", null, null,
                                  false );
```

```
while( msgEnum.hasMoreElements() )
{
    /* get message */
    MQeMsgObject msg = msgEnum.getNextMessage( null, MQe.uniqueValue() );
    /* confirm get */
    qmgr.confirmGetMessage( msgEnum.getQueueManagerName(),
                           msgEnum.getQueueName(), msg.getMsgUIDFields() ); }
    ...
}
```

Related functions

getQueueManagerName

MQeException

MQeException

This class is used to create an MQeException object.

Package **com.ibm.mqe**

This class is a descendant of **MQe**

Constructor summary

Constructor	Purpose
MQeException	Constructs an MQeException object.

Method summary

Method	Purpose
code	Returns the integer value of the exception

MQeException

Syntax

1. `public MQeException()`
2. `public MQeException(int codeValue)`
3. `public MQeException(String errorMsg)`
4. `public MQeException(int codeValue, String errorMsg)`

Description

Constructs an MQeException object. There are five forms of constructor:

1. Creates an object that has a **codeValue** of 0 and does not have an error message
2. Creates an object that has the specified **codeValue** and does not have an error message
3. Creates an object that has a **codeValue** of 0 and has an error message
4.
 - a. Creates an object that has the specified **codeValue** and has an error message
 - b. Creates an object that has the specified **codeValue**, has an error message and has imbedded (hidden) data

The value of the **codeValue** parameter should be one of the constants defined in the **MQe** class, for example, **MQe.Except_NotFound**.

Parameters

- codeValue** An integer value, normally one of the MQe.Except_... constants
- errorMsg** A String associated with the exception, and possibly displayed when the exception occurs

Return values

none

Exceptions

none

Example

```

class MySampleClass
{
    ...
    if ( data == null )
        throw new MQeException( MQe.Except_Data, "Data missing" );
    ...
    ...
}

```

MQeException code**Syntax**

```
public int code( )
```

Description

This method extracts the code value of the MQeException, that is the value that was set when the exception was created.

Parameters

none

Return values

An integer

Exceptions

none

Example

```

class MySampleClass
{
    ...
    try
    {
        ...
    }
    catch ( Exception e )
    {
        if ( e instanceof MQeException )
            switch (((MQeException) e).code( ) )
            {
                case MQe.Except_Data:
                    System.err.println( "Data format error" );
                    break;
                case MQe. Except_NotFound:
                    System.err.println( "Data not specified" );
                    break;
            }
        else
            System.err.println( "Error:" + e.toString( ) );
        ...
    }
}

```

MQeFields

This class is used to create a basic MQeFields object. This object is used to hold various data items and provide mechanisms to dump and restore these field items to or from a byte array.

Fields items are assigned a character name at the time they are added to the Fields object. This name must:

- Be at least 1 character long
- Conform to the ASCII character set, that is characters with values $20 < \text{value} < 128$
- Must not include any of the characters `{ } [] # () ; ; ' " =`

Note: These rules are not enforced but the results are unpredictable if they are not followed.

Package **com.ibm.mqe**

This class is a descendant of **MQe**

Constructor summary

Constructor	Purpose
MQeFields	Creates and initializes theMQeFields object

Method summary

Method	Purpose
contains	Verifies that the field exists within the object
copy	Copies a field or set of fields from one MQeFields object to another
dataType	Determines the data type of a field in the object
delete	Removes a field from the object
dump	Dumps the contents of the message object to a byte array
dumpedType	Returns the object type of the dumped Fields object
dumpToString	Produces a human readable representation of the contents of the Fields object
equals	Performs an equality test with another Fields object
fields	Returns an enumeration of all the fields in the object
getArrayLength	Extracts the length value of a dynamic array of field's
getArrayOfByte	Extracts a fixed size array of bytes
getArrayOfDouble	Extracts a fixed size array of double size floating point numbers
getArrayOfFloat	Extracts a fixed size array of float size floating point numbers
getArrayOfInt	Extracts a fixed size array of int size integers
getArrayOfLong	Extracts a fixed size array of long size integers
getArrayOfShort	Extracts a fixed size array of short size integers
getAscii	Extracts an Ascii string
getAsciiArray	Extracts an Ascii array of strings

Method	Purpose
getAttribute	Extracts the current attribute object reference
getBoolean	Extracts a boolean value or <i>null</i>
getByte	Extracts a byte value
getByteArray	Extracts a dynamic size array of byte values
getDouble	Extracts a double floating point value
getDoubleArray	Extracts a dynamic size array of double floating point values
getFields	Extracts an imbedded Fields object
getFieldsArray	Extracts a dynamic size array of Fields objects
getFloat	Extracts a float value
getInt	Extracts an integer
getIntArray	Extracts a dynamic size integer array
getLong	Extracts a long integer
getLongArray	Extracts a dynamic size long integer array
getShort	Extracts a short integer
getShortArray	Extracts a short integer array
getUnicode	Extracts a Unicode string
getUnicodeArray	Extracts a dynamic size array of Unicode strings
hide	Prevents a field from being used within the equals check
putArrayLength	Sets the length value of a dynamic array of field's
putArrayOfByte	Sets a fixed size array of bytes
putArrayOfDouble	Sets a fixed size array of double size floating point numbers
putArrayOfFloat	Sets a fixed size array of float size floating point numbers
putArrayOfInt	Sets a fixed size array of int size integers
putArrayOfLong	Sets a fixed size array of long size integers
putArrayOfShort	Sets a fixed size array of short size integers
putAscii	Sets a string containing ASCII characters
putAsciiArray	Sets a dynamic size array of strings containing ASCII characters
putBoolean	Sets a boolean value
putByte	Sets in the message data from a byte
putByteArray	Sets a dynamic size array of byte values
putDouble	Sets a double floating point value
putDoubleArray	Sets a dynamic size double floating-point array
putFields	Sets an imbedded Fields object
putFieldsArray	Sets an array of Fields objects
putFloat	Sets a float value
putFloatArray	Sets a dynamic size float array
putInt	Sets an integer
putIntArray	Sets a dynamic size integer array
putLong	Sets a long integer
putLongArray	Sets a dynamic size long integer array
putShort	Sets a short integer

MQeFields

Method	Purpose
putShortArray	Sets a dynamic size short integer array
putUnicode	Sets a string containing Unicode characters
putUnicodeArray	Sets a dynamic size array of strings containing Unicode characters
rename	Renames an item held within the fields object
restore	Restores the contents of a Fields object from a byte array produced by the dump method
restoreFromFile	Restores the contents of a fields object from a file in either binary or formatted ASCII
restoreFromString	Restores the contents of a Fields object from an ascii string (typically produced by a dumpToString method call)
setAttribute	Assigns an attribute object to this MQeFields
updateValue	Updates (increments or decrements) an integer type value within a Fields object

MQeFields

Syntax

1. `public MQeFields()`
2. `public MQeFields(byte data[])`

Description

The constructor creates and initializes the MQeFields object. There are two forms of the constructor:

1. With no parameters. This constructs an empty Fields object.
2. With a byte array. This restores a Fields object from the supplied byte array.

Note: The objects must be of the same type

Parameters

data A byte array containing a dumped Fields object

Return values

none

Exceptions

MQException Except_data, "data:xxxx"
Except_Type, "Type: aaaa - bbbb"

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    ...
    ...
}
```

MQeFields contains

Syntax

```
public boolean contains( String item )
```

Description

This method verifies that a field exists within the MQeFields object

Parameters

item The name of the item to be checked

Return values

true the field was found
false the field was not found

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields msg = new MQeFields( );
    msg.putAscii("Data", "This is some data" );
    ...
    if ( msg.contains( "Data" ) )
        ...
    ...
}
```

MQeFields copy**Syntax**

1. public void copy(MQeFields from,
 boolean replace)
2. public void copy(MQeFields from,
 boolean replace,
 String item)

Description

This method will copy a reference to a field (or all field's) from one Fields object to another. There are two forms:

1. Copies all fields
2. Copies an individual field

The boolean value **replace**, if set to **false**, throws an exception if the field already exists within the target Fields object, if set to **true**, it replaces the value that already exists.

Parameters

from The MQeFields object to be used as the source of the data
replace A boolean controlling if a field is to be replaced or not
item The name of a single field to be copied

Return values

none

Exceptions

MQeException Except_Duplicate, "Duplicate: aaaa"

Example

```
class MyApplication
{
    ...
}
```

MQeFields

```
MQeFields fields1 = new MQeFields( );
fields1.putAscii("data", "This issome data" );
...
MQeFields fields2 = new MQeFields( );
fields2.copy(fields1, true, "data" )
...
...
}
```

MQeFields dataType

Syntax

```
public char dataType( String item )
```

Description

This method will return the data type of a field within the MQeFields object.

Parameters

item The name of the item to be checked

Return values

A character value representing the data type of the field. The predefined data types in MQeFields are:

```
public    final static char    TypeUnTyped
public    final static char    TypeAscii
public    final static char    TypeUnicode
public    final static char    TypeBoolean
public    final static char    TypeByte
public    final static char    TypeShort
public    final static char    TypeInt
public    final static char    TypeLong
public    final static char    TypeFloat
public    final static char    TypeDouble
public    final static char    TypeArrayElements
public    final static char    TypeFields
```

Exceptions

MQeException Various

Example

```
class MyApplication
{
...
MQeFieldsmsg = new MQeFields( );
msg.putAscii("Data", "This is some data" );
...
if ( msg.dataType( "Data" ) ==TypeInt
...
}
```

MQeFields delete

Syntax

```
public void delete( String item )
```

Description

This method deletes an existing field from the MQeFields object.

Parameters

item The name of the item to be removed

Return values**Exceptions**

none

Example

```

class MyApplication
{
    ...
    MQeFields msg = new MQeFields( );
    msg.putAscii("Data", "This is some data" );
    ...
    msg.delete( "Data" );
    ...
}

```

MQeFields dump**Syntax**

1. public byte[] dump() throws Exception
2. public byte[] dump(boolean allowXor) throws Exception

Description

This method dumps out to a byte array, the contents of this MQeFields object so that it can be restored using the restore method. There are two forms of this method:

1. With no parameter
2. With **allowXor**. If this is set to **false**, the MQeFields object is dumped to a byte array. With **allowXor** set to **true**, each field is XOR'd with a previous version (held internally) in an attempt to increase the number of bytes that have a value of 0x00 to try to improve the compression ratio.

If intelligent fields objects are to be created, that is fields that have program logic in them, then it is in the dump and restore methods that this logic should be activated. For example when just before the fields object is dumped, a data base query could be issued to get latest data, or just after a restore data could be automatically stored in a data base.

Parameters

allowXor A boolean expression. **true** implies XOR the Fields, **false** implies do not XOR the Fields

Return values

none

Exceptions

MQeException Various

Example

```

class MyApplication
{
    ...
    MQeFields msg = new MQeFields( );
    msg.putAscii( "Data", "This is some data" );
    ...
    byte dumpData[] = msg.dump( );
    ...
}

```

MQeFields dump data format

Data sent between MQSeries Everyplace environments is encoded using the following layout:

```
{Length Identifier Fence {Data}} {Length Identifier Fence {Data}} { ... }
```

Where:

Length

A variable number of bytes between 1 and 4. The length is encoded in the following manner:

The first byte has the first two bits reserved and they are used as the length of the length field:

- 00 = 1 byte used for length (6 bits = 0-63)
- 01 = 2 bytes used for length (14 bits = 0-16,383)
- 10 = 3 bytes used for length (22 bits = 0-4,194,303)
- 11 = 4 bytes used for length (30 bits = 0-1,073,741,823)

Identifier

A variable length string of bytes (each byte value must be less than 0x80) typically this would be an ASCII string. The end of the identifier is determined when a byte with 0xC0 bits set, is encountered. The identifier is subject to the following restrictions:

- Must be at least 1 character long
- Conform the ASCII character set (20<value<128)
- Must not include any of the characters {}[]#()';'=""

Fence A special byte delimiting the boundary between the identifier and the optional data item. This byte is used to contain the data type of the data item as shown in the following example:

```
/* Field mask values */
public final static char TypeFenceMask = 0x00C0;
public final static char TypeHidden = 0x0020;
public final static char TypeModifier = 0x0010;
/* Field data types */
public final static char TypeUnTyped = 0x0000 | TypeFenceMask;
public final static char TypeAscii = 0x0001 | TypeFenceMask;
public final static char TypeUnicode = 0x0002 | TypeFenceMask;
public final static char TypeBoolean = 0x0003 | TypeFenceMask;
public final static char TypeByte = 0x0004 | TypeFenceMask;
public final static char TypeShort = 0x0005 | TypeFenceMask;
public final static char TypeInt = 0x0006 | TypeFenceMask;
public final static char TypeLong = 0x0007 | TypeFenceMask;
public final static char TypeFloat = 0x0008 | TypeFenceMask;
public final static char TypeDouble = 0x0009 | TypeFenceMask;
public final static char TypeArrayElements = 0x000A | TypeFenceMask;
public final static char TypeFields = 0x000B | TypeFenceMask;
```

The order of the items within the data stream is not significant.

```
08 5349 C7 1122334455 |02 44 D3 |03 5349 D6 |4603 534443 C4
6E01534FE32054E16....
```

Savings in number of bytes transmitted

Using this data structure savings in the byte stream are achieved by:

- Reserving two bits in the first length byte allowing variable length lengths. Variable length Length code (1 to 4 bytes) i.e. only the required length bytes are sent.

- Leading 0x00s and 0xFF's of integer values are not placed in the output stream. If a value is "0" or "-1" no data bytes are sent.
- All the data items are typed and can be type checked at the receiving end.
- Null items are still transmitted (with data type), hence the presence of the item can be checked at the receiving end.
- Using the Fence byte for 3 distinct functions
 1. Delimiting the Identifier
 2. Defining the data type
 3. Defining that the data is:
 - *null* (no Data bytes)
 - positive or negative (possibly no Data bytes sent, 0 or -1)
 - boolean true or false (no Data bytes sent)

Note: Further savings can be achieved by compressing the data. The compressors can usually be helped by performing an XOR with a previous byte stream producing repeated bytes of 0x00, but because of the variable nature of these fields and the order of the fields may change a simple XOR will not produce the desired effect. However an "intelligent" XOR that worked on a field by field basis produces repeated 0x00 bytes thus assisting the compressor

MQeFields dumpToString

Syntax

```
public String dumpToString( String template )
```

Description

This method dumps the Fields object in human readable form and returns the data as a String.

Parameters

- template** A String template used when formatting the output. The template should have 3 insert sequences '#n'. That is:
- "#0" for the data type,
 - "#1" for the Field name
 - "#2" for the field value.

Example:

```
"Sample template -Name=#1, Type=#0, Value=#2"
```

Return values

A String containing a representation of the Fields object

Exceptions

Various conversion Exceptions

Examples

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.putBoolean( "tb", true );
    ...
    fields.putLong( "m1", -1 );
    System.out.println( fields.dumpToString( "Test1.obj  (#0)\t#1\t=#2\r\n" ) );
    ...
}
```

MQeFields

Example output from the dumpToString call:

```
Test1.obj (long)    la  =[2] { 0000000000000001, FFFFFFFF }
Test1.obj (boolean) tb  =true Test1.obj (byte) ba =[5] { 01, FE, FD, 04, 05 }
Test1.obj (long)    pl  =101
Test1.obj (ascii)   A   =Ascii string
Test1.obj (ascii)   nA  =null
Test1.obj (unicode) U   =Unicode string
Test1.obj (byte)    mb  =[1] { FE }
Test1.obj (int)     i   =1
Test1.obj (byte)    pb  =[1] { 02 }
Test1.obj (boolean) fb  =false
Test1.obj (short)   ms  =-1
Test1.obj (short)   sa  =[5] { 0001, FFFE, FFFD, 0004, 0005}
Test1.obj (short)   ps  =0
Test1.obj (int)     ia  =[3] { 00000001, FFFFFFFE, FFFFFFFD }
Test1.obj (long)    ml  =-1
```

MQeFields dumpedType

Syntax

```
public static String dumpedType( byte data[] ) throws Exception
```

Description

This method returns a String containing the class name of the object that was dumped.

Parameters

data A byte array containing a dump of a fields object

Return values

A String containing the class name of the object that was dumped

Exceptions

MQeException Except_Data,, "Data:aaa"

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.putAscii( "Data", "This is some data" );
    byte dumpdata[] = fields.dump(fields.dump( );
    ...
    String ObjType = fields.dumpedType(objType = MQeFields.dumpedType( dumpdata );
    ...
}
```

MQeFields equals

Syntax

```
public boolean equals( MQeFields match ) throws Exception
```

Description

The default method requires an MQeFields (or descendent) as a parameter, each field in the parameter object is checked for equality with a matching field in the MQeFields.

Override this method to provide a different type of equality check.

Parameters

match An MQeFields object containing the items to be used for the comparison.

Return values**true** if there is a match otherwise **false****Exceptions**

MQeException	Except_Type, "wrong field type"
MQeException	Except_NotFound, item + " not found"

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.putAscii( "Data1", "This is some data" );
    fields.putAscii( "Data2", "This is more data" );
    ...
    MQeFields test = new MQeFields( );
    test.putAscii( "Data1", "This is some data" );
    ...
    if ( fields.equals( test ) )
        ...
    else
        ...
}
```

MQeFields fields**Syntax**

```
public Enumeration fields( )
```

Description

This method returns an enumeration object that contains all the field names within the object.

Parameters

none

Return values

An enumeration object containing the field names

Exceptions

MQeException	Except_Type, "wrong field type"
MQeException	Except_NotFound, item + " not found"

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.putAscii( "data", "This is some data" );
    ...
    Enumeration names = fields.fields( );
    ...
}
```

MQeFields getArrayLength**Syntax**

```
public int getArrayLength( String item ) throws Exception
```

MQeFields

Description

This extracts the dynamic array length of the specified item. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved

Return values

An array of strings containing the ASCII data from the message

Exceptions

MQeException Except_Type, "wrong field type"
Except_NotFound, item + " not found"

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( dumpData );
    ...
    int numElements = fields.getArrayLength( "Data" );
    ...
}
```

MQeFields getArrayOfByte

Syntax

```
public byte[] getArrayOfByte( String item ) throws Exception
```

Description

This extracts an array of bytes data from the Fields object. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved

Return values

A byte array containing the data from the message

Exceptions

MQeException Except_Type, "wrong field type"
Except_NotFound, item + " not found"

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.restore( dumpData );
    ...
    byte data[] = fields.getArrayOfByte( "Data" );
    ...
}
```

MQeFields getArrayOfDouble

Syntax

```
public double[] getArrayOfDouble( String item ) throws Exception
```

Description

This extracts an array of double floating point numbers from the Fields object. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved

Return values

An array of double values

Exceptions

MQeException Except_Type, "wrong field type"
 Except_NotFound, item + " not found"

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.restore( dumpData );
    ...
    double data[] = fields.getArrayOfDouble( "Data" );
    ...
}
```

MQeFields getArrayOfFloat**Syntax**

```
public float[] getArrayOfFloat( String item ) throws Exception
```

Description

This extracts an array of floating point numbers from the Fields object. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved

Return values

An array of float values

Exceptions

MQeException Except_Type, "wrong field type"
 Except_NotFound, item + " not found"

Example

```
class MyApplication
{
    ...

    MQeFields fields = new MQeFields( );
    fields.restore( dumpData );
    ...
    float data[] = fields.getArrayOfFloat( "Data" );
    ...
}
```

MQeFields

MQeFields getArrayOfInt

Syntax

```
public int[] getArrayOfInt( String item ) throws Exception
```

Description

This extracts an array of int length integer numbers from the Fields object. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved.

Return values

An array of int values

Exceptions

MQeException	Except_Type, "wrong field type"
	Except_NotFound, item + " not found"

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.restore( dumpData );
    ...
    int data[] = fields.getArrayOfInt( "Data" );
    ...
}
```

MQeFields getArrayOfLong

Syntax

```
public long[] getArrayOfLong( String item ) throws Exception
```

Description

This extracts an array of long length integer numbers from the Fields object. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved

Return values

An array of long values

Exceptions

MQeException	Except_Type, "wrong field type"
	Except_NotFound, item + " not found"

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.restore( dumpData );
    ...
    long data[] = fields.getArrayOfLong( "Data" );
    ...
}
```

MQeFields getArrayOfShort

Syntax

```
public short[] getArrayOfShort( String item ) throws Exception
```

Description

This extracts an array of short length integer numbers from the Fields object. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved

Return values

An array of short values

Exceptions

MQException	Except_Type, "wrong field type"
	Except_NotFound, item + " not found"

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.restore( dumpData );
    ...
    short data[] = fields.getArrayOfShort( "Data" );
    ...
}
```

MQeFields getAscii

Syntax

```
public String getAscii( String item ) throws Exception
```

Description

This extracts the ASCII data from the Fields object and returns it as a string. An exception is thrown if there is no data or it is of the wrong data type.

Note: The `item` parameter is a java Unicode string, which must only contain character codes that appear in the invariant part of the Ascii code pages (characters with values $20 < \text{value} < 128$, not including `{[]#():;'"=}`). If you attempt to pass variant character codes, these codes are subject to translations between machines when different codepages are used to manipulate the data, possibly resulting in unpredictable results. If you wish to pass variant character codes in an MQSeries Everyplace message, we recommend you use the `putArrayOfByte()` method and handle your own codepage translations between machines, or `putUnicode()` method where no codepage translations is required.

Parameters

item The name of the item to be retrieved

Return values

A string containing the ASCII data from the message

Exceptions

MQeFields

MQException Except_Type, "wrong field type"
 Except_NotFound, item + " not found"

Example

```
class MyApplication
{
...
MQeFields fields = new MQeFields();
fields.restore( dumpData );
...
String data = fields.getAscii( "Data" );
...
}
```

MQeFields getAsciiArray

Syntax

```
public String[] getAsciiArray( String item ) throws Exception
```

Description

This extracts the ASCII data (see note in "MQeFields getAscii" on page 85) from the Fields object and returns it as an array of strings. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved

Return values

An array of strings containing the ASCII data from the message

Exceptions

MQException Except_Type, "wrong field type"
 Except_NotFound, item + " not found"

Example

```
class MyApplication
{
...
MQeFields fields = new MQeFields( );
fields.restore( dumpData );
...
String data[] = fields.getAsciiArray( "Data" );
...
}
```

MQeFields getAttribute

Syntax

```
public MQeAttribute getAttribute( )
```

Description

This method returns an MQeAttribute object reference associated with this Fields object, or *null* if there is no attribute.

Parameters

none

Return values

An MQeAttribute object reference

Exceptions

none

Example

```
class MyApplication
{
...
MQeAttribute thisAttribute = fields.getAttribute( );
...
}
```

MQeFields getBoolean**Syntax**

```
public boolean getBooean( String item ) throws Exception
```

Description

This extracts a boolean value from the Fields object. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved

Return values

A boolean set to either **true** or **false**

Exceptions

MQeException	Except_Type, "wrong field type"
	Except_NotFound, item + " not found"

Example

```
class MyApplication
{
...
MQeFields fields = new MQeFields( );
fields.restore( dumpData );
...
boolean data = fields.getBoolean( "Data" );
...
}
```

MQeFields getByte**Syntax**

```
public byte getByte( String item ) throws Exception
```

Description

This extracts a byte of data from the Fields object. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved

Return values

A byte containing the data from the field

Exceptions

MQeException	Except_Type, "wrong field type"
	Except_NotFound, item + " not found"

MQeFields

Example

```
class MyApplication
{
...
MQeFields fields = new MQeFields( );
fields.restore( dumpData );
...
byte data = fields.getBytes( "Data" );
...
}
```

MQeFields getByteArray

Syntax

```
public byte[] getByteArray( String item ) throws Exception
```

Description

This extracts the byte data from the Fields object and returns it as a byte array. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved

Return values

A byte array containing the data from the field

Exceptions

MQException	Except_Type, "wrong field type"
	Except_NotFound, item + " not found"

Example

```
class MyApplication
{
...
MQeFields fields = new MQeFields( );
fields.restore( dumpData );
...
byte data[] = fields.getByteArray( "Data" );
...
}
```

MQeFields getDouble

Syntax

```
public double getDouble( String item ) throws Exception
```

Description

This extracts a double length floating point value from the Fields object. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved

Return values

A double containing the value from the field

Exceptions

MQException	Except_Type, "wrong field type"
	Except_NotFound, item + " not found"

Example

```

class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.restore( dumpData );
    ...
    double data = fields.getDouble( "Data" );
    ...
}

```

MQeFields getDoubleArray**Syntax**

```
public byte[] getDoubleArray( String item ) throws Exception
```

Description

This extracts a dynamic array of double length floating point values from the Fields object and returns it as an array of doubles. The length of the array is determined by the ArrayLength value for this item. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved

Return values

A byte array containing the data from the field

Exceptions

MQeException	Except_Type, "wrong field type"
	Except_NotFound, item + " not found"

Example

```

class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.restore( dumpData );
    ...
    double data[] = fields.getDoubleArray( "Data" );
    ...
}

```

MQeFields getFields**Syntax**

```
public MQeFields getFields( String item ) throws Exception
```

Description

This extracts a Fields object item from the Fields object. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved

Return values

A double containing the value from the field

Exceptions

MQeFields

MQeException	Except_Type, "wrong field type"
	Except_NotFound, item + " not found"

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.restore( dumpData );
    ...
    MQeFields data = fields.getFields( "Data" );
    ...
}
```

MQeFields getFieldsArray

Syntax

```
public MQeFields[] getFieldsArray( String item ) throws Exception
```

Description

This extracts a dynamic array of MQeFields objects from the Fields object and returns it as an array. The length of the array is determined by the ArrayLength value for this item. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item	The name of the item to be retrieved
-------------	--------------------------------------

Return values

An array containing the MQeFields objects

Exceptions

MQeException	Except_Type, "wrong field type"
	Except_NotFound, item + " not found"

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.restore( dumpData );
    ...
    MQeFields data[] = fields.getFieldsArray( "Data" );
    ...
}
```

MQeFields getFloat

Syntax

```
public float getFloat( String item ) throws Exception
```

Description

This extracts a float value item from the Fields object. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item	The name of the item to be retrieved
-------------	--------------------------------------

Return values

A float containing the value from the field

Exceptions

MQeException	Except_Type, "wrong field type"
	Except_NotFound, item + " not found"

Example

```
class MyApplication
{
...
MQeFields fields = new MQeFields( );
fields.restore( dumpData );
...
float data = fields.getFloat( "Data" );
...
}
```

MQeFields getFloatArray**Syntax**

```
public float[] getFloatArray( String item ) throws Exception
```

Description

This extracts a dynamic array of float values from the Fields object and returns it as an array. The length of the array is determined by the ArrayLength value for this item. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved

Return values

An array containing the float values

Exceptions

MQeException	Except_Type, "wrong field type"
	Except_NotFound, item + " not found"

Example

```
class MyApplication
{
...
MQeFields fields = new MQeFields( );
fields.restore( dumpData );
...
float data[] = fields.getFloatArray( "Data" );
...
}
```

MQeFields getInt**Syntax**

```
public int getInt( String item ) throws Exception
```

Description

This extracts an int length integer value item from the Fields object. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved

MQeFields

Return values

An int containing the value from the field

Exceptions

MQeException	Except_Type, "wrong field type"
	Except_NotFound, item + " not found"

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.restore( dumpData );
    ...
    int data = fields.getInt( "Data" );
    ...
}
```

MQeFields getIntArray

Syntax

```
public int[] getIntArray( String item ) throws Exception
```

Description

This extracts a dynamic array of int length integer values from the Fields object and returns it as an array. The length of the array is determined by the ArrayLength value for this item. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item	The name of the item to be retrieved
-------------	--------------------------------------

Return values

An array containing the int values

Exceptions

MQeException	Except_Type, "wrong field type"
	Except_NotFound, item + " not found"

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.restore( dumpdDta );
    ...
    int data[] = fields.getIntArray( "Data" );
    ...
}
```

MQeFields getLong

Syntax

```
public long getLong( String item ) throws Exception
```

Description

This extracts an long length integer value item from the Fields object. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved

Return values

A long containing the value from the field

Exceptions

MQeException	Except_Type, "wrong field type"
	Except_NotFound, item + " not found"

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.restore( dumpData );
    ...
    long data = fields.getLong( "Data" );
    ...
}
```

MQeFields getLongArray

Syntax

```
public long[] getLongArray( String item ) throws Exception
```

Description

This extracts a dynamic array of long length integer values from the Fields object and returns it as an array. The length of the array is determined by the ArrayLength value for this item. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved

Return values

An array containing the long values

Exceptions

MQeException	Except_Type, "wrong field type"
	Except_NotFound, item + " not found"

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.restore( dumpData );
    ...
    long data[] = fields.getLongArray( "Data" );
    ...
}
```

MQeFields getShort

Syntax

```
public short getShort( String item ) throws Exception
```

Description

This extracts a short length integer value item from the Fields object. An exception is thrown if there is no data or it is of the wrong data type.

MQeFields

Parameters

item The name of the item to be retrieved

Return values

A short integer containing the value from the field

Exceptions

MQeException Except_Type, "wrong field type"
 Except_NotFound, item + " not found"

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.restore( dumpData );
    ...
    short data = fields.getShort( "Data" );
    ...
}
```

MQeFields getShortArray

Syntax

```
public short[] getShortArray( String item ) throws Exception
```

Description

This extracts a dynamic array of short length integer values from the Fields object and returns it as an array. The length of the array is determined by the ArrayLength value for this item. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved

Return values

An array containing the short values

Exceptions

MQeException Except_Type, "wrong field type"
 Except_NotFound, item + " not found"

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.restore( dumpData );
    ...
    short data[] = fields.getShortArray( "Data" );
    ...
}
```

MQeFields getUnicode

Syntax

```
public String getUnicode( String item ) throws Exception
```

Description

This extracts the Unicode data from the message object and returns it as a string. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved

Return values

A string containing the Unicode data from the message

Exceptions

MQeException	Except_Type, "wrong field type"
	Except_NotFound, item + " not found"

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.restore( dumpData );
    ...
    String data = fields.getUnicode( "Data" );
    ...
}
```

MQeFields getUnicodeArray**Syntax**

```
public String[] getUnicodeArray( String item ) throws Exception
```

Description

This extracts the Unicode data from the message object and returns it as an array of strings. An exception is thrown if there is no data or it is of the wrong data type.

Parameters

item The name of the item to be retrieved

Return values

An array of strings containing the Unicode data from the message

Exceptions

MQeException	Except_Type, "wrong field type"
	Except_NotFound, item + " not found"

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.restore( dumpData );
    ...
    String data[] = fields.getUnicodeArray( "Data" );
    ...
}
```

MQeFields

MQeFields hide

Syntax

```
public void hide( String item, boolean state ) throws Exception
```

Description

This sets an item within the Fields object to be included (**true**) or not included (**false**), when an equals test is performed against the Fields object

Parameters

item	The name of the item to be hidden/included.
state	hide (true) or include (false)

Return values

none

Exceptions

MQeException	Except_NotFound, item + " not found"
---------------------	--------------------------------------

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( dumpData );
    ...
    fields.hide( "Data" );
    if ( OldFields.equals( fields ) )
        ...
    ...
}
```

MQeFields putArrayLength

Syntax

```
public void putArrayLength( String item, int length ) throws Exception
```

Description

This sets the dynamic array length of the specified item.

Parameters

item	The name of the item to be set
length	The length of the array in number of elements

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( dumpData );
    ...
    fields.putArrayLength( "Data", 5 );
    ...
}
```


MQeFields putArrayOfByte

Syntax

```
public void putArrayOfByte( String item, byte data ) throws Exception
```

Description

This method sets the data in the message object for the supplied byte array.

Parameters

item	The name of the item to be set
data	A byte array containing the data to be set into the message object

Return values

none

Exceptions

MQeException	Various
---------------------	---------

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    ...
    fields.putArrayOfByte( "Data", new byte[] { 1, 2, 3, 4 } );
    ...
}
```

MQeFields putArrayOfDouble

Syntax

```
public void putArrayOfDouble( String item, double data[] )
                               throws Exception
```

Description

This sets an array of double floating point numbers into the Fields object.

Parameters

item	The name of the item to be set
data	The array of double values to be copied into the Fields object

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.restore( dumpData );
    ...
    double data[] = fields.putArrayOfDouble( "Data" );
    ...
}
```

MQeFields

MQeFields putArrayOfFloat

Syntax

```
public void putArrayOfFloat( String item, float data[] )  
                                throws Exception
```

Description

This sets an array of floating point numbers into the Fields object.

Parameters

item	The name of the item to be set
data	The array of float values to be copied into the Fields object

Return values

none

Exceptions

none

Example

```
class MyApplication  
{  
    ...  
    MQeFields fields = new MQeFields( );  
    fields.restore( dumpData );  
    ...  
    float data[] = fields.putArrayOfFloat( "Data" );  
    ...  
}
```

MQeFields putArrayOfInt

Syntax

```
public void putArrayOfInt( String item, int data[] ) throws Exception
```

Description

This sets an array of int length integer numbers into the Fields object.

Parameters

item	The name of the item to be set
data	The array of int values to be copied into the Fields object

Return values

none

Exceptions

none

Example

```
class MyApplication  
{  
    ...  
    MQeFields fields = new MQeFields();  
    fields.restore( dumpData );  
    ...  
    fields.putArrayOfInt( "Data",new int[] { 1, 2, 3, 4 } );  
    ...  
}
```

MQeFields putArrayOfLong

Syntax

```
public void putArrayOfLong( String item, long data[] ) throws Exception
```

Description

This sets an array of long length integer numbers into the Fields object.

Parameters

item	The name of the item to be set
data	The array of long values to be copied into the Fields object

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields();
    fields.restore( dumpData );
    ...
    fields.putArrayOfLong( "Data",new long[] { 1, 2, 3, 4 } );
    ...
}
```

MQeFields putArrayOfShort

Syntax

```
public void putArrayOfShort( String item, short data[] )
    throws Exception
```

Description

This sets an array of short length integer numbers into the Fields object.

Parameters

item	The name of the item to be retrieved
data	The array of long values to be copied into the Fields object

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.restore( dumpData );
    ...
    fields.putArrayOfShort( "Data", new short[] { 1, 2, 3, 4 } );
    ...
}
```

MQeFields

MQeFields putAscii

Syntax

```
public void putAscii( String item, String data ) throws Exception
```

Description

This method sets ASCII data (see note in "MQeFields getAscii" on page 85) into the Fields object and sets the data type.

Parameters

item	The name of the item to be set
data	A string containing the data to be set into the Fields object

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    ...
    fields.putAscii( "Data", "This is some data" );
    ...
}
```

MQeFields putAsciiArray

Syntax

```
public void putAsciiArray( String item, String data[] ) throws Exception
```

Description

This method sets ASCII data (see note in "MQeFields getAscii" on page 85) from an array of strings into the Fields object and sets the data type.

Parameters

item	The name of the item to be set
data	A string array containing the data to be set into the Fields object

Return values

none

Exceptions

MQeException	Various
---------------------	---------

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    ...
    String data[] = { "This is some data", "This is more data" };
    fields.putAsciiArray( "Data", data[] );
    ...
}
```

MQeFields putBoolean

Syntax

```
public void putBoolean( String item, boolean data ) throws Exception
```

Description

This sets a boolean value into the Fields object.

Parameters

item	The name of the item to be set
data	The boolean value to be set

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.restore( dumpData );
    ...
    fields.putBoolean( "Data", false );
    ...
}
```

MQeFields putByte

Syntax

```
public void putByte( String item, byte data ) throws Exception
```

Description

This method sets the data in the Fields object for the supplied byte.

Parameters

item	The name of the item to be set
data	A byte containing the data to be set into the message object

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    ...
    fields.putByte( "Data", 123 );
    ...
}
```

MQeFields putByteArray

Syntax

```
public void putByteArray( String item, byte data[] ) throws Exception
```

MQeFields

Description

This method sets the data in the Fields object for the supplied array of byte arrays.

Parameters

item	The name of the item to be set
data	An array of byte arrays containing the data to be set into the Fields object

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    ...
    byte data[][] = new byte[2][];
    data[1][] = { 1, 2, 3, 4 };
    data[2][] = { 5, 6, 7, 8 };
    fields.putByteArray( "Data", data );
    ...
}
```

MQeFields putDouble

Syntax

```
public void putDouble( String item, double data ) throws Exception
```

Description

This method sets the data in the Fields object for the supplied double value.

Parameters

item	The name of the item to be set
data	A double value to be set into the Fields object

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    ...
    fields.putDouble( "Data", 123.456 );
    ...
}
```

MQeFields putDoubleArray

Syntax

```
public void putDoubleArray( String item, byte data[][] )
                        throws Exception
```

Description

This method sets an array of double length floating point values into the Fields.

Parameters

item The name of the item to be set
data An array of double values to be set into the Fields object

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    ...
    double data[] = new byte[2];
    data[1] = 1.234;
    data[2] = 5.678;
    fields.putDoubleArray( "Data", data );
    ...
}
```

MQeFields putFields**Syntax**

```
public void putFields( String item, MQeFields data ) throws Exception
```

Description

This method sets the data Fields object as an item within this Fields object.

Parameters

item The name of the item to be set
data An MQeFields object to be set into this Fields object

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    ...
    MQeFields subFields = new MQeFields( );
    ...
    fields.putFields( "Data", subFields );
    ...
}
```

MQeFields

MQeFields putFieldsArray

Syntax

```
public void putFieldsArray( String item, MQeFields data[] )  
                           throws Exception
```

Description

This method sets an array of MQeFields objects into this Fields.

Parameters

item	The name of the item to be set
data	An array of MQeFields objects to be set into this Fields object

Return values

none

Exceptions

none

Example

```
class MyApplication  
{  
    ...  
    MQeFields fields = new MQeFields( );  
    ...  
    MQeFields subFields = new MQeFields[2];  
    MQeFields subFields[0] = new MQeFields( );  
    MQeFields subFields[1] = new MQeFields( );  
    ...  
    fields.putFieldsArray( "Data", subFields );  
    ...  
}
```

MQeFields putFloat

Syntax

```
public void putFloat( String item, float data ) throws Exception
```

Description

This method sets the data in the Fields object for the supplied float value.

Parameters

item	The name of the item to be set
data	A float value to be set into the Fields object

Return values

none

Exceptions

none

Example

```
class MyApplication  
{  
    ...  
    MQeFields fields = new MQeFields( );  
    ...  
    fields.putFloat( "Data", 123.456 );  
    ...  
}
```


MQeFields putFloatArray

Syntax

```
public void putFloatArray( String item, float data[] ) throws Exception
```

Description

This method sets an array of floating point values into the Fields object.

Parameters

item	The name of the item to be set
data	An array of floating point values to be set into the Fields object

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    ...
    float data[] = new float[2];
    data[1] = 1.234;    data[2] = 5.678;
    fields.putFloatArray( "Data", data );
    ...
}
```

MQeFields putInt

Syntax

```
public void putInt( String item, int data ) throws Exception
```

Description

This method sets the an int length integer into the Fields object.

Parameters

item	The name of the item to be set
data	An int value to be set into the Fields object

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    ...
    fields.putInt( "Data", 123456 );
    ...
}
```

MQeFields

MQeFields putIntArray

Syntax

```
public void putIntArray( String item, int data[] ) throws Exception
```

Description

This method sets an array of int length integer values into the Fields object.

Parameters

item	The name of the item to be set
data	An array of int values to be set into the Fields object

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    ...
    int data[] = new int[2];
    data[1] = 1234; data[2] = 5678;
    fields.putIntArray( "Data", data );
    ...
}
```

MQeFields putLong

Syntax

```
public void putLong( String item, long data ) throws Exception
```

Description

This method sets the a long length integer into the Fields object.

Parameters

item	The name of the item to be set
data	A long value to be set into the Fields object

Return values

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    ...
    fields.putLong( "Data", 123456 );
    ...
}
```

MQeFields putLongArray

Syntax

```
public void putLongArray( String item, long data[] ) throws Exception
```

Description

This method sets an array of long length integer values into the Fields object.

Parameters

item	The name of the item to be set
data	An array of long values to be set into the Fields object

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    ...
    long data[] = new long[2];
    data[1] = 1234; data[2] = 5678;
    fields.putLongArray( "Data", data );
    ...
}
```

MQeFields putShort

Syntax

```
public void putShort( String item, short data ) throws Exception
```

Description

This method sets the a short length integer into the Fields object.

Parameters

item	The name of the item to be set
data	A short value to be set into the Fields object

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    ...
    fields.putShort( "Data", 123 );
    ...
}
```

MQeFields

MQeFields putShortArray

Syntax

```
public void putShortArray( String item, short data[] ) throws Exception
```

Description

This method sets an array of short length integer values into the Fields object.

Parameters

item	The name of the item to be set
data	An array of short values to be set into the Fields object

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    ...
    short data[] = new short[2];
    data[1] = 1234;
    data[2] = 5678;
    fields.putShortArray( "Data", data );
    ...
}
```

MQeFields putUnicode

Syntax

```
public void putUnicode( String item, String data ) throws Exception
```

Description

This method sets Unicode data into the message object and sets the data type.

Parameters

item	The name of the item to be set
data	A string containing the data to be set into the message object

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields msg = new MQeFields( );
    ...
    msg.putUnicode( "Data", "Merry xmas to all our readers" );
    ...
}
```

MQeFields putUnicodeArray

Syntax

```
public void putUnicodeArray( String item, String data[] )
                           throws Exception
```

Description

This method sets Unicode data into the message object for the string array and sets the data type.

Parameters

item	The name of the item to be set
data	An array of strings containing the data to be set into the message object

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
    MQeFields msg = new MQeFields( );
    ...
    String data[] = new String[2];
    data[1] = "Merry xmas to all our readers";
    data[2] = "and a happy new year";
    msg.putUnicode( "Data", data );
    ...
}
```

MQeFields rename

Syntax

```
public void rename( String itemName, String newName ) throws Exception
```

Description

This method renames an existing item within the fields object to the specified new name. If an item already exists in the fields object with the **newName** it is replaced by the renamed item..

Parameters

itemName	A String containing the name of the item to be renamed.
newName	A String containing the new name of the item

Return values

none

Exceptions

MQeException	Except_NotFound, Item + " not found"
---------------------	--------------------------------------

Example

```
class MyApplication
{
    ...
    dumpDatafields.rename( "ThisItem", "ThatItem" );
    ...
}
```

MQeFields

MQeFields restore

Syntax

```
public void restore( byte data[] ) throws Exception
```

Description

This method restores a message object from a byte array that was created using the dump method.

Parameters

data A byte array containing a dumped MQeFields

Return values

none

Exceptions

MQeException	Except_Type, "wrong field type"
	Except_NotFound, item + " not found"
	Except_data, "data:xxxx"
	Except_Type, "Type: aaaa - bbbb"

Example

```
class MyApplication
{
...
MQeFields fields = new MQeFields( );
...
fields.restore( dumpData );
...
}
```

MQeFields restoreFromFile

Syntax

1. public MQeFields restoreFromFile(String fileName, MQeAttribute attribute) throws Exception
2. public MQeFields restoreFromFile(String fileName, String endRecord, String sectionMatch, String template) throws Exception

Description

These methods create a new MQeFields object from the contents of a disk file. There are two forms:

1. A binary file containing a byte array from a dumped MQeFields object
2. An ASCII file containing sections delimited by records delimited by endRecord strings and nested fields within fields object from a sectionMatch and endRecord delimited string

Parameters

fileName	A String containing the name of the file to be read
attribute	An MQeAttribute object used to decode (decrypt and/or decompress) the binary data
endRecord	A String containing the characters that delimit an end of record. E.g. '\r\n'
sectionMatch	A String containing a pattern template for:

- The section name, for example '[#0]'
- The characters that delimit an end of record, for example '\r\n'

The sectionMatch template should have only one insert sequence '#0'

template A String template used to parse the input.

The Template should have up to 3 insert sequences '#n':

#0" for the data type

#1" for the Field name

#2" for the field value

as shown in the following example:

Name=#1, Type=#0, Value=#2

Return values

An MQeFields object containing the restored values

Exceptions

Various conversion exceptions

Example

```
class MyApplication
{
...
MQeFields fields = MQeFields.restoreFromFile(File.separator + "directory" +
File.separator + "thisfile.xyz",
"\r\n",
"[#0]",
"#1=#2" );
...
}
```

The preceding example processes an ASCII file with the following structure

```
[Section1]
item1=1235678
item2=abcdef
[Section2]
item1=qwertyiop
```

It constructs a fields object containing two imbedded fields objects with item names of **Section1** and **Section2**, each of these imbedded fields has the relevant items from that section.

MQeFields restoreFromString

Syntax

1. public void restoreFromString(String template, String data) throws Exception
2. public void restoreFromString(String endRecord, String template, String data) throws Exception
3. public static MQeFields restoreFromString(String endRecord, String sectionMatch, String template, String data) throws Exception

Description

These methods restore:

MQeFields

1. An individual item from string
2. A group of items from an EndRecord delimited string
3. A nested Fields within Fields object from a SectionMatch and EndRecord delimited string

Parameters

template

A String template used to parse the input.

The Template should have up to 3 insert sequences "#n":

"#0" for the data type

"#1" for the Field name

"#2" for the field value

as shown in the following example:

Name=#1, Type=#0, Value=#2

Return values

An MQeFields object containing the restored values

Exceptions

Various conversion exceptions

Example

```
class MyApplication
{
    ...
    MQeFields fields = new MQeFields( );
    fields.putBoolean( "tb", true );
    ...
    fields.putLong( "m1", -1 );
    String data = fields.dumpToString( "Name=#1, Type=#0 Value=#2\r\n" );
    ...
    MQeFields newFields = new MQeFields( );
    newFields.restoreFromString( "Name=#1, Type=#0 Value=#2\r\n", data );
    ...
}
```

MQeFields setAttribute

Syntax

```
public void setAttribute( MQeAttribute attribute ) throws Exception
```

Description

This method assigns an attribute to be used to encode or decode the contents of the Fields object when ever it is dumped or restored.

Parameters

attribute An MQeAttribute object reference

Return values

none

Exceptions

none

Example

```
class MyApplication
{
    ...
```



```

MQeFields fields = new MQeFields( );
...
MQeAttribute attr = new MQeAttribute( null, new MQeXorCryptor( ), null );
fields.setAttribute( attr );
...
}

```

MQeFields updateValue

Syntax

```
public long updateValue( String item, long update ) throws Exception
```

Description

This method increments or decrements an integer value held within this Fields object.

Parameters

item	The name of the item to be set
update	A value to be added to the current value of the specified item

Return values

The updated value

Exceptions

MQeException	Various
---------------------	---------

Example

```

class MyApplication
{
...
MQeFields fields = new MQeFields( );
...
fields.putInt( "Data", 123 );
...
long l = fields.updateValue("Data", -3 );
...
}

```

MQeKey

This class is used to create an MQeKey object. MQeKey objects can be attached to and used by an attribute object. Attributes are associated with channel and MQeFields objects.

Package **com.ibm.mqe**

This class is a descendant of **MQe**

Constructor summary

Constructor	Purpose
MQeKey	Constructs an MQeKey object

Method summary

Method	Purpose
setLocalKey	Sets local encrypt and decrypt keys to a value derived from the CipherKey seed provided

MQeKey

Syntax

```
public MQeKey( )
```

Description

Constructs an MQeKey object

Parameters

none

Return values

none

Exceptions

none

Related functions

- MQeAttribute
- MQeMAttribute

MQeKey setLocalKey

Syntax

```
Public void setLocalKey ( String localCipherKey ) throws MQeException
```

Description

Protects data and writes it to the given target filename

Parameters

localCipherKey

Seed from which the Key encrypt and decrypt keys are derived

Return Values

none

Exceptions**MQeException**Except_NotAllowed, "invalid
localCipherKey"**Example**

```

class MySampleClass extends MQe
{
  try
  {
    /* protecting MQeFields data */
    MQeDESCryptor des = new MQeDESCryptor( );
    MQeAttribute desA = new MQeAttribute( null, des, null);
    MQeKey localkey = new MQeKey();
    localkey.setLocalKey( "It_is_a_secret");
    desA.setKey( localkey );
    MQeFields localf = new MQeFields( );
    localf.setAttribute( desA );
    Trace ( "i: test data = " + "0123456789abcdef...." );
    localf.putArrayOfByte(
      "TestData", asciiToByte("0123456789abcdef....") );
    byte[] temp = localf.dump( );
    Trace ( "i: test data protected using MQeKey = " +
      byteToHex( temp ) );

    /* unprotecting MQeFields data */
    MQeDESCryptor des2 = new MQeDESCryptor( );
    MQeAttribute desA2 = new MQeAttribute( null, des2, null);
    MQeKey localkey2 = new MQeKey();
    localkey2.setLocalKey( "It_is_a_secret");
    desA2.setKey( localkey );
    MQeFields localf2 = new MQeFields( );
    localf2.setAttribute( desA2 );
    localf2.restore ( temp );
    Trace ( "i: unprotected test data = " +
      byteToAscii(localf2.getArrayOfByte( "TestData" ) );
  }
  catch ( Exception e )
  {
  }
}

```

Related functions

- **MQeLocalSecure**

MQeMessageEvent

This object is passed to an application when an MQSeries Everyplace Message event occurs.

Package **com.ibm.mqe**

extends **java.util.EventObject**

Method summary

Method	Purpose
getMsgFields	Returns an MQeFields object containing selected fields from the message that caused the event to be generated.
getQueueManagerName	Returns a String containing the name of the queue manager that owns the queue that generated this event
getQueueName	Returns a String containing the name of the queue that generated this event

MQeMessageEvent getMsgFields

Syntax

```
public MQeFields getMsgFields()
```

Description

This method returns an MQeFields object containing selected fields from the message that caused the event to be generated. The unique id of the message (consisting of a timestamp plus the origin queue manager name) is always returned together with the MQSeries Message Id, MQSeries Correlation Id, and message priority values if they are present in the message.

Parameters

none.

Return values

An MQeFields object containing selected fields from the message that caused the event to be generated.

Exceptions

none.

Example

```
class MyMQeApplication
{
    ...
    /* called when a msg event occurs */
    public void messageArrived( MQeMessageEvent e )
    {
        String eventQueueName = e.getQueueName(); /* get origin Q name */
        if ( eventQueueName.equals( "SYSTEM.DEFAULT.LOCAL.QUEUE" ) )
        {
            ...
            /* get msg info */
            MQeFields filter = e.getMsgFields();
            System.out.println( "Message received from QueueMgr: " +
                e.getQueueManagerName() );
            qmgr.getMessage( null, "SYSTEM.DEFAULT.LOCAL.QUEUE", filter, null, 0 );
            ...
        }
    }
}
```

```

    } ...
} ...

```

MQeMessageEvent getQueueManagerName

Syntax

```
public String getQueueManagerName()
```

Description

This method returns a String containing the name of the queue manager that owns the queue that generated this event.

Parameters

none

Return values

A String containing the name of the queue manager that owns the queue that generated this event.

Exceptions

none

Example

```

class MyMQeApplication
{
    ...
    /* called when a msg event occurs */
    public void messageArrived( MQeMessageEvent e )
    {
        String eventQMgr = e.getQueueManagerName(); /* get origin QMgr */
        String eventQueueName = e.getQueueName(); /* get origin Q name */

        if ( eventQMgr.equals( localQMgr.getName() ) )
        { /* local QMgr */
            ...
        }
        ...
    }
    ...
}

```

Related functions

[getQueueName](#)

MQeMessageEvent getQueueName

Syntax

```
public String getQueueName()
```

Description

This method returns a String containing the name of the queue that generated this event.

Parameters

none

Return values

A String containing the name of the queue that generated this event.

Exceptions

none

Example

MQeMessageEvent

```
class MyMQApplication
{
    ...
    /* called when a msg event occurs */
    public void messageArrived( MQeMessageEvent e )
    {
        String eventQueueName = e.getQueueName(); /* get origin Q name */
        if ( eventQueueName.equals( "SYSTEM.DEFAULT.LOCAL.QUEUE" ) )
        {
            ...
        }
        ...
        if ( eventQueueName.equals( "MyQueue" ) )
        {
            ...
        }
    }
    ...
}
```

Related functions
getQueueManagerName

MQeMsgObject class

This section describes the Java class used to create a basic MQeMsgObject, this object is used to hold data, or to contain the necessary logic to obtain the data to send from one MQSeries Everyplace system to another. Normally a descendant of this class would be used to hold additional characteristics, data or code.

Package **com.ibm.MQe**

This class is a descendant of **MQeFields**

Constants and variables

The following field name constants from the MQSeries Everyplace base class are used by MQeMsgObject.

The following two constants combine to make up the unique message identifier. They are set by the MQSeries Everyplace system and the application should not attempt to modify these values. **Msg_OriginQMgr** is an ASCII item and **Msg_Time** is a long integer value.

```
public final static String  Msg_OriginQMgr
public final static String  Msg_Time
```

The following field name constants are provided for use by message destined for or received from MQS systems and are not required for messages wholly within an MQSeries Everyplace environment.

They should be treated as byte arrays, and if they are to sent to MQS then they should be 24 bytes in length.

```
public final static String  Msg_CorrelID
public final static String  Msg_MsgID
```

The following field name constant is provided for use by message destined for or received from MQS systems and is not required for messages wholly within an MQSeries Everyplace environment. It is used to set the message style.

```
public final static String  Msg_Style
```

This constant is an int integer and may have the following values:

```
public final static int     Msg_Style_Datagram
public final static int     Msg_Style_Request
public final static int     Msg_Style_Reply
```

The following field name constants are provided for use by a message destined for, or received from, an MQS system and are not required for messages wholly within an MQSeries Everyplace environment. They can be used to have equal meaning by an application retrieving messages from a queue in an MQSeries Everyplace environment.

Both these field items are ASCII.

```
public final static String  Msg_ReplyToQ
public final static String  Msg_ReplyToQMgr
```

The following field name constant must be a byte value between the 0 and 9 and it sets the message priority. If it is not set when added to a queue then the queues default priority value takes effect.

```
public final static String  Msg_Priority
```

MQeMsgObject

The following field name constant can be used to expire a message. MQe can discard the message if it has expired.

This constant can have one of two meanings.

1. If it is a long integer, the expire time is considered as the absolute time and date after which the message can be discarded.
2. If the item is an int integer, the expire time is relative to the creation time of the message object.

```
public final static String  Msg_ExpireTime
```

The following field name is a Boolean value that the application can set or can be set by the system to indicate that the message is or has been resent.

This resend flag is set and/or reset by the MQSeries Everyplace system in order to control guaranteed message delivery on it's internal flows.

```
public final static String  Msg_Resend
```

The following field name constant is a long integer value, normally the value returned by a browseMessages request when browsing with lock. It is not required for messages wholly within an MQSeries Everyplace environment.

```
public final static String  Msg_LockID
```

Constructor summary

Constructor	Purpose
MQeMsgObject	Creates and initializes the MQeMsgObject object

Method summary

Method	Purpose
getMsgUIDFields	Extracts the unique identifier for the message
getOriginQMgr	Extracts the name of the origination queue manager (if present)
getTimeStamp	Extracts the time the message object was created
putOriginQMgr	Sets the messages originating queue manager name Note: Once set this cannot be changed.
resetMsgUIDFields	Resets the message objects Unique id
unwrapMessageObject	Unwraps a wrapped message object

MQeMsgObject

Syntax

1. public MQeMsgObject()
2. public MQeMsgObject (byte data[])
3. public MQeMsgObject (MQeMsgObject msg)

Description

The constructor creates and initializes the MQeMsgObject object. There are three forms of the constructor:

1. With no parameters. This constructs an empty message object.
2. With a byte array. This restores a fields object from the supplied byte array .

Note: The objects must be of the same type

3. With a MQeMsgObject. This wrappers the supplied message into a new message object. This is normally used to wrapper messages that have attributes attached that force the data to be held encoded.

Parameters

data A byte array containing a dumped fields object.

Return values

none

Exceptions

MQeException	Except_Type, "wrong field type"
	Except_NotFound, item + " not found"
	Except_data, "data:xxxx"
	Except_Type, "Type: aaaa - bbbb"

Example

```
class MyApplication
{
...
MQeMsgObject msg = new MsgObject( );
...
...
}
```

MQeMsgObject getMsgUIDFields

Syntax

```
public MQeFields getMsgUIDFields ( )
```

Description

This method returns an **MQeFields** object containing the following field items:

msg_Time	The time the message was created
msg_OriginQMGr	The name of the originating queue manager

The returned fields object can be used as a match parameter in an equality test or in a browse or get message call.

Parameters

none

Return values

An **MQeFields** object containing the fields used to make the unique message identifier.

Exceptions

MQeException	Except_NotAllowed, 'Queue Manager not set'
---------------------	--

Example

```
class MyApplication
{
...
...
}
```

MQeMsgObject

```
...
MQeFields uid = Msg.getMsgIDFields( );
...
}
```

MQeMsgObject getOriginQMgr

Syntax

```
public String getOriginQMgr( )
```

Description

Returns a String containing the name of the originating queue manager or *null* if not set.

Parameters

none

Return values

A String or *null*

Exceptions

none

Example

```
class MyApplication
{
...
...
String uid = Msg.getOriginQMgr();
...
}
```

MQeMsgObject getTimeStamp

Syntax

```
public long getTimeStamp( )
```

Description

Returns a long integer value containing the time in milliseconds when the object was created.

Parameters

none

Return values

A long value in milliseconds

Exceptions

none

Example

```
class MyApplication
{
...
...
long uid = Msg.getTimeStamp ( );
...
}
```

MQeMsgObject putOriginQMgr

Syntax

```
public void putOriginQMgr( )
```

Description

Sets the name of the originating queue manager. Once this name has been set, it cannot be reset.

Note: Normally this method would only be called internally by the queue manager when PutMessage call is issued.

Parameters

none

Return values

none

Exceptions

none

MQeMsgObject resetMsgUIDFields**Syntax**

```
public void resetMsgUIDFields ( )
```

Description

This method resets the message objects unique id such that a new **Msg_Time** value is generated and the **Msg_OriginQMgr** is set to *null*. This in effect creates a new message object but retains any field items that were set.

Parameters

none

Return values

none

Exceptions

none

Example

```
{
...
...
msg.resetMsgUIDFields ( );
...
}
```

MQeMsgObject unwrapMessageObject**Syntax**

```
public MQeMsgObject unwrapMessageObject( MQeAttribute attribute )
```

Description

This method unwraps an imbedded MQeMsgObject, decode using the supplied attribute (if appropriate), and returns the new message object.

Parameters

attribute An **MQeAttribute** object reference or *null*, used to decode the imbedded message object

Return values

none

Exceptions

none

MQeMsgObject

Example

```
class MyApplication
{
    ...
    ...
    msg.resetMsgUIDFields( );
    ...
}
```

MQeQueue

MQeQueue is the base queue class, all other types of queue are descendants of this object.

queues are objects that hold messages, the messages are held in a message store owned by the queue. Typically, this message store would be a persistent storage device, such as a hard disk. However, other types of store can be used, such as a database. MQSeries Everyplace relies on the fact that the queues have persistent store to be able to offer it's assured message delivery, and so the use of any non-persistent storage, would invalidate MQe's assured message delivery.

The queue uses a *queue store adapter* to handle its communications with the storage device. Adapters are interfaces between MQSeries Everyplace and hardware devices, such as disks or networks, or to software, such as databases. Adapters are designed to be pluggable components, allowing the queue store to be easily changed.

The messages held on the queue can be protected by an authenticator and cryptor. The messages can also be compressed by using a compressor. Together, the authenticator, cryptor, and compressor are known as the attributes of the queue, and they are defined by specifying an appropriate MQeAttribute object to be associated with the queue.

The behaviour of the queue is governed by a set of rules. These rules take the form of a Java class and can be extended by an MQSeries Everyplace solution. The base set of queue rules is defined in the class, **MQeQueueRule**. During the operation of the queue, the rules are called when certain events occur, for example, when a message is put, a message expires, or a duplicate message arrives. The rules then determine how the queue handles these events.

A queue expiry interval can be defined. If a message has remained on the queue for a length of time greater than the queue's expiry interval, then the message is marked as expired. The queue's rules then determine what happens to the message, but typically the message would be either deleted, or placed on a 'dead letter queue'. The queue's expiry interval is different to a message's expiry interval.

The maximum number of messages and the maximum allowable size of an individual message can also be defined.

All queues are owned by a queue manager. The queues owned by a queue manager are known as that queue manager's *local queues*. A queue manager can also access queues belonging to another queue manager. These queues are known as *remote queues*. When a queue manager accesses a remote queue, it stores the information it has learnt about the characteristics of that queue. The information is stored in a remote queue definition. The remote queue definition is represented by the class, **MQeRemoteQueue**.

Package
com.ibm.mqe

Method summary

Method	Purpose
<code>getCreationDate</code>	This method returns a long value representing the date and time at which the queue was created.

MQeQueue

Method	Purpose
<code>getDefaultPriority</code>	This method returns an integer value that is the queue's default priority.
<code>getDescription</code>	This method returns a String object containing the description string for the queue.
<code>getExpiryInterval</code>	This method returns a long value that is the message expiry interval (in milliseconds) for the queue.
<code>getMaxMessageSize</code>	This method returns an integer value that is the maximum size of message (in bytes) which the queue can hold.
<code>getMaxQueueSize</code>	This method returns an integer value that is the maximum number of messages that can be held on this queue.
<code>getNumberOfMessages</code>	This method returns an integer value that is the current number of messages held on this queue.
<code>getQueueAttribute</code>	This method returns a MQeAttribute object, which defines the authenticator, cryptor, and compressor used by this queue.
<code>getQueueName</code>	This method returns a String object containing the name of the queue.
<code>getQueueStore</code>	This method returns a String object containing the pathname of the queue's persistent store.

MQeQueue getCreationDate

Syntax

```
public long getCreationDate()
```

Description

This method returns the date and time that this queue was created.

Parameters

none

Return values

A long value representing the time that this queue was created

Exceptions

none

Example

```
class ExampleQueueManagerRules extends MQeQueueManagerRule
{
    ...
    public void addQueue( MQeQueue queue ) throws MQeException
    {
        /* only allow the addition of queues created after 1st June 2000 */
        Calendar calendar = Calendar.getInstance();
        calendar.set( 2000, 05, 01 );
        Date date = calendar.getTime();
        /* get current time */
        Date qDate = new Date( queue.getCreationDate() );
        /* compare the two dates */
        if ( date.after( qDate ) )
            throw new MQeException( Except_Rule, "addQueue disallowed" );
    }
    ...
}
```

MQeQueue getDefaultPriority

Syntax

```
public int getDefaultPriority()
```

Description

This method returns the Queue's default priority value. This is the priority value which will be used for any message placed on the Queue that has not previously been assigned a priority value.

Parameters

none

Return Values

An integer value which is the default priority for this queue.

Exceptions

MQeException	none
---------------------	------

Example

```
class ExampleQueueManagerRules extends MQeQueueManagerRule
{
    ...
    public boolean transmit( MQeQueue queue )
    {
        /* allow transmission if queue's priority is greater than 5 */
        if ( queue.getDefaultPriority() > 5 )
            return (true);
        else
            return (false);
    }
    ...
}
```

MQeQueue getDescription

Syntax

```
public String getDescription()
```

Description

This method returns the description string for this queue.

Parameters

none

Return values

A String object containing the description string for this queue.

Exceptions

none

Example

```
import examples.eventlog.*;

class ExampleQueueManagerRules extends MQeQueueManagerRule
{
    public final static int Event_QueueManager_AddQueue = 201;
    LogToDiskFile logFile = null;
    ...
    public void addQueue( MQeQueue queue )
    {
        /* log the addition of a new queue */
        log( MQe_Log_Information, Event_QueueManager_AddQueue,
            "Queue " + queue.getQueueManagerName() + "+" + queue.getQueueName() +
```

MQeQueue

```
        ": " + queue.getDescription() );
    }

    public void queueManagerActivate() throws Exception
    {
        /* create a new log file */
        logFile = new LogToDiskFile( "\\log.txt");
    }

    public void queueManagerClose()
    {
        /* close log file */
        logFile.close();
    }
    ...
}
```

MQeQueue getExpiryInterval

Syntax

```
public long getExpiryInterval()
```

Description

This method returns the message expiry interval for this queue. Any message that has been on the queue for a length of time greater than the expiry interval will be marked as expired. The queue's rules then determine what happens to the message.

Parameters

none

Return Values

A long value that is the message expiry interval (in milliseconds) for this queue. A value of zero means that the queue has no message expiry interval set.

Exceptions

none

Example

```
class ExampleQueueManagerRules extends MQeQueueManagerRule
{
    ...
    public boolean transmit( MQeQueue queue )
    {
        /* transmit if queue has a low message expiry time (less than 1 day) */
        /* (zero means no expiry) */
        if ( queue.getExpiryInterval() < (60 * 60 * 24 * 1000) &&
            queue.getExpiryInterval() > 0 )
            return (true);
        else
            return (false);
    }
    ...
}
```

MQeQueue getMaxMessageSize

Syntax

```
public int getMaxMessageSize()
```

Description

This method returns the maximum size of a message (in bytes) that can be held on this queue.

Parameters

none

Return Values

An integer value that is the maximum size of a message (in bytes) that can be held on this queue.

Exceptions

none

Example

```

{
    ...
    public void addQueue( MQeQueue queue ) throws MQeException
    {
        /* only allow addition of queue if it supports messages of at least 2MB */
        if ( queue.getMaxMessageSize() < 2048000 )
            throw new MQeException( Except_Rule, "Message size too small" );
    }
    ...
}

```

MQeQueue getMaxQueueSize**Syntax**

```
public int getMaxQueueSize()
```

Description .

This method returns the maximum number of messages that can be held on this queue.

Parameters

none

Return Values

An integer value that is the maximum number of messages that can be held on this queue.

Exceptions

none

Example

```

class ExampleQueueManagerRules extends MQeQueueManagerRule
{
    ...
    public void addQueue( MQeQueue queue ) throws MQeException
    {
        /* only allow addition of queue if it supports more than 100 messages */
        if ( queue.getMaxQueueSize() < 100 )
            throw new MQeException( Except_Rule, "Max Queue depth too small" );
    }
    ...
}

```

MQeQueue getNumberOfMessages**Syntax**

```
public int getNumberOfMessages()
```

Description

This method returns the current number of messages held on this queue.

Parameters

none

MQeQueue

Return Values

An integer value that is the current number of messages held on this queue.

Exceptions

none

Example

```
class ExampleQueueManagerRules extends MQeQueueManagerRule
{
    ...
    public boolean transmit( MQeQueue queue )
    {
        /* only allow queue to transmit if it contains more than 10 messages */
        if ( queue.getNumberOfMessages() >= 10 )
            return (true);
        else
            return (false);
    }
    ...
}
```

MQeQueue getQueueAttribute

Syntax

```
public MQeAttribute getQueueAttribute()
```

Description

This method returns this queue's attribute object. The attribute object defines the authenticator, cryptor and compressor used by this queue. These attributes are used upon any messages stored on the queue.

Parameters

none

Return Values

A MQeAttribute object that defines the authenticator, cryptor, and compressor used by the queue.

Exceptions

none

Example

```
class ExampleQueueManagerRules extends MQeQueueManagerRule
{
    ...
    public void addQueue( MQeQueue queue ) throws Exception
    {
        /* only allow addition of queues with a defined DES Cryptor */
        MQeAttribute qAttribute = queue.getQueueAttribute();
        if ( qAttribute == null )
            throw new MQeException( Except_Rule, "No queue attribute defined" );

        MQeCryptor cryptor = qAttribute.getCryptor();
        if ( cryptor == null )
            throw new MQeException( Except_Rule, "No cryptor defined" );

        if ( !(cryptor.securityLevel().equals( "DES" )) )
            throw new MQeException( Except_Rule, "DES Cryptor not defined" );
    }
    ...
}
```

MQeQueue getQueueManagerName

Syntax

```
public String getQueueManagerName()
```

Description

This method returns the name of the Queue Manager to which this Queue belongs.

Parameters

none

Return Values

A String object containing the name of the Queue Manager to which this Queue belongs.

Exceptions

none

Examples

```
import examples.eventlog.*;

class ExampleQueueManagerRules extends MQeQueueManagerRule
{
    public final static int Event_QueueManager_AddQueue = 201;
    LogToDiskFile logFile = null;
    ...
    public void addQueue( MQeQueue queue )
    {
        /* log the addition of a new queue */
        log( MQe_Log_Information, Event_QueueManager_AddQueue,
            "Queue " + queue.getQueueManagerName() + "+" + queue.getQueueName() +
            ": " + queue.getDescription() );
    }

    public void queueManagerActivate() throws Exception
    {
        /* create a new log file */
        logFile = new LogToDiskFile( "\\log.txt");
    }

    public void queueManagerClose()
    {
        /* close log file */
        logFile.close();
    }
    ...
}
```

Related Functions

[getQueueName](#)

MQeQueue getQueueName

Syntax

```
public String getQueueName()
```

Description

This method returns the name of this queue.

Parameters

none

Return values

A String object that is the name of this queue.

MQueQueue

Exceptions

none

Examples

```
import examples.eventlog.*;

class ExampleQueueManagerRules extends MQueQueueManagerRule
{
    public final static int Event_QueueManager_AddQueue = 201;
    LogToDiskFile logFile = null;
    ...
    public void addQueue( MQueQueue queue )
    {
        /* log the addition of a new queue */
        log( MQue_Log_Information, Event_QueueManager_AddQueue,
            "Queue " + queue.getQueueManagerName() + "+" + queue.getQueueName() +
            ": " + queue.getDescription() );
    }

    public void queueManagerActivate() throws Exception
    {
        /* create a new log file */
        logFile = new LogToDiskFile( "\\log.txt");
    }

    public void queueManagerClose()
    {
        /* close log file */
        logFile.close();
    }
    ...
}
```

Related functions

[getQueueManagerName](#)

MQueQueue [getQueueStore](#)

Syntax

```
public String getQueueStore()
```

Description

This method returns the pathname to the queue's persistent store.

Parameters

none

Return values

A String object that is the pathname to the queue's persistent store.

Exceptions

none

Example

```
import examples.eventlog.*;

class ExampleQueueManagerRules extends MQueQueueManagerRule
{
    public final static int Event_QueueManager_AddQueue = 201;
    LogToDiskFile logFile = null;
    ...
    public void addQueue( MQueQueue queue )
    {
        /* log the addition of a new queue */
        log( MQue_Log_Information, Event_QueueManager_AddQueue,
            "Queue " + queue.getQueueManagerName() + "+" + queue.getQueueName() +
```

```
        ": " + queue.getDescription() + ": Persistent store: " +
        queue.getQueueStore() );
    }

    public void queueManagerActivate() throws Exception
    {
        /* create a new log file */
        logFile = new LogToDiskFile( "\\log.txt");
    }

    public void queueManagerClose()
    {
        /* close log file */
        logFile.close();
    }
    ...
}
```

MQeQueueManager

This class is used to construct an MQSeries Everyplace queue manager object.

The MQSeries Everyplace queue manager is the focal point of the MQSeries Everyplace system. It provides:

- A central point of access to the MQSeries Everyplace and MQS network for MQSeries Everyplace applications
- Once-only assured delivery of messages
- Full recovery from failure conditions
- Extendable rules-based behavior

Package

`com.ibm.mqe`

Constructor summary

Constructor	Purpose
<code>MQeQueueManager</code>	Constructs a <code>MQeQueueManager</code> object.

Method summary

Method	Purpose
<code>activate</code>	Activates a queue manager that has already been instantiated.
<code>addMessageListener</code>	Registers an object as a listener for MQSeries Everyplace message events.
<code>browseMessages</code>	Returns an <code>MQeEnumeration</code> containing the messages on the specified queue that match the specified filter.
<code>browseMessagesAndLock</code>	Works identically to <code>browseMessages</code> but with the addition that all messages returned are left locked on the queue.
<code>checkActive</code>	Returns a boolean value denoting whether or not the queue manager is active.
<code>close</code>	Closes down the queue manager.
<code>confirmGetMessage</code>	Confirms a previous <code>getMessage</code> operation.
<code>confirmPutMessage</code>	Confirms a previous <code>putMessage</code> operation.
<code>deleteMessage</code>	Removes a message from the specified queue.
<code>getMessage</code>	Returns a message from the specified queue.
<code>getName</code>	Returns the unique queue manager name.
<code>getReference</code>	Returns an object reference to the specified queue manager.
<code>putMessage</code>	Places a message onto the specified queue.
<code>removeMessageListener</code>	Cancel an objects subscription to MQSeries Everyplace message events.
<code>triggerTransmission</code>	Allows an application to initiate the transmission of any pending messages.
<code>undo</code>	This method is intended to be used in the event of an error whilst executing a <code>putMessage()</code> , <code>getMessage()</code> or <code>browseMessagesAndLock()</code> command.

Method	Purpose
<code>unlockMessage</code>	Unlocks a message that has previously been locked by a <code>browseMessagesAndLock</code> operation.
<code>waitForMessage</code>	Performs an identical function to <code>getMessage</code> , with the exception that if no message is available, the queue manager waits for a specified period for a message to become available.

MQQueueManager

Syntax

1. `public MQQueueManager()`
2. `public MQQueueManager(MQFields startupParameters) throws Exception`

Description

Constructs an **MQQueueManager** object. The **activate()** method must be called to initialize and start the queue manager.

There are two versions of the constructor:

1. Creates a default queue manager that must be initialized by calling the `activate` method.
2. Creates and starts a new queue manager using the parameters supplied in the specified `MQFields` object. Subsequently the queue manager is activated via an internal call to the `activate` method.

Note: Before starting a queue manager, all required `MQSeries` Everyplace aliases must have been added. Refer to the *MQSeries Everyplace for Multiplatforms Programming Guide* for details of how to do this.

Parameters

startupParameters

An `MQFields` object containing the startup parameters for the queue manager.

The startup parameters must contain two sections, **MQQueueManager.QueueManager**, which sets up the queue manager and **MQQueueManager.Registry**, which sets up the registry.

MQQueueManager.QueueManager

This section contains the following:

MQQueueManager.Name

An ascii string containing the queue manager's unique name. This name must:

- Be at least 1 character long
- Conform to the ASCII character set, i.e. characters with values $20 < \text{value} < 128$
- Must not include any of the characters `{ } [] # () ; , ' " =`
- The first character of the queue manager name should be alphanumeric

However, to maintain compatibility with `MQSeries`, it is recommended that queue manager names are

MQeQueueManager

limited to a maximum length of 48 characters. The characters can be any of the following:

- Uppercase A-Z
- Lowercase a-z · Numerics 0-9
- Period (.)
- Underscore (_)
- Forward slash (/)
- Percent sign (%)

MQeQueueManager.Registry

This registry must exist, and must contain the following:

MQeRegistry.LocalRegType

An ascii string containing the type of registry to use. Currently the only recognized types are MQeRegistry.FileRegistry and MQeRegistry.PrivateRegistry.

Note: Once a registry has been created, it is recommended that the registry type is not changed. Changing the registry type may cause secure queues to function incorrectly

MQeRegistry.DirName

An ascii string containing a pathname to the queue manager's registry.

Two further sections,

MQeQueueManager.AppRunList, and **MQeQueueManager.CloseAppRunList** are optional. These sections specify a list of MQSeries Everyplace applications that are invoked once the queue manager is active, and when it receives a close request. (See "MQeRunListInterface" on page 202

Return values

none

Exceptions

MQeException

Except_QMgr_Activated

Except_QMgr_AlreadyExists

Except_QMgr_InvalidQMgrName

Except_QMgr_NotConfigured

Examples

```
import com.ibm.mqe.*;
public class StartQueueManager {
public static void main(String[] args) {
try {
if (null == args || args.length < 1 || null == args[0]) {
System.out.println("No ini file name supplied");
} else {
```


MQeQueueManager

```
System.out.println("Reading ini file: " + args[0]);
MQeFields iniSections = MQeFields.restoreFromFile(args[0],
                                                "\r\n", "[#0]",
                                                "(#0)#1=#2");

MQeQueueManager queueManager = new MQeQueueManager();
System.out.println("Activating queueManager");
queueManager.activate(iniSections);
System.out.println("Closing queueManager");
queueManager.close();
System.out.println("Closed queueManager");
}
} catch (Exception e) {
System.out.println("Yet another exception thrown: " + e.getMessage());
e.printStackTrace();
}
}
```

MQeQueueManager activate

Syntax

1. public void activate(MQeFields startupParameters) throws Exception
2. public void activate(String name) throws Exception

Description

Note: Before starting a queue manager, all required MQSeries Everyplace aliases must have already been added. Refer to the MQSeries Everyplace Programmers Guide for details of how to do this.

There are two versions of the method:

1. This is the recommended version of the method. It takes as input an **MQeFields** object containing startup parameters for the queue manager. The queue manager then initializes all its subcomponents correctly, and reads any information stored in its registry.
2. This version is provided solely to allow the standard queue manager activation procedure to be overridden by a class extending from **MQeQueueManager**. This method performs no activation procedure other than to set the queue manager name.

Note: Any class extending **MQeQueueManager** must call **MQeQueueManager.activate()** to ensure that the queue manager name is set correctly.

Parameters

startupParameters

An **MQeFields** object containing startup parameters for the queue manager.

See the **MQeQueueManager startupParameters** for details of these parameters.

name

An ascii string containing the name of the queue manager

Return values

none

Exceptions

MQeException

Except_QMgr_Activated

MQeQueueManager

Except_QMgr_AlreadyExists
Except_QMgr_InvalidQMgrName
Except_QMgr_NotConfigured
Except_NotFound

Example

```
class MyMQeApplication
{
    ...
    /* Create QueueManager startup parameters */
    MQeFields QMgrParams = new MQeFields();
    QMgrParams.PutAscii( MQeQueueManager.Name, "TestQMgr" );
    QMgrParams.PutAscii( MQeQueueManager.QueueStore, "MsgLog:c:\\TestQMgr" );

    /* Create Registry startup parameters */
    MQeFields RegParams = new MQeFields();
    RegParams.PutAscii( MQeQueueManager.RegType, MQeQueueManager.FileRegistry );
    RegParams.PutAscii( MQeQueueManager.Path, "MsgLog:c:\\TestQMgr\\Registry" );

    /* Combine the two sets of parameters into a single Fields object */
    MQeFields Params = new MQeFields();
    Params.PutFields( MQeQueueManager.QueueManager, QMgrParams );
    Params.PutFields( MQeQueueManager.Registry, RegParams );

    /* Instantiate 'null' Queue Manager */
    MQeQueueManager QMgr = new MQeQueueManager( );
    QMgr.Activate( Params ); /* Activate QMgr using parameters */
    ...
}
```

Related functions

- close

MQeQueueManager addMessageListener

Syntax

```
public void addMessageListener( MQeMessageListenerInterface listener,
                               String queueName,
                               MQeFields filter ) throws Exception;
```

Description

This method registers an object as a listener to any MQeMessage events generated by the queue specified in the **queueName** parameter. It is only possible to add listeners to local queues.

Note: The listening object must implement the **MQeMessageListenerInterface**. Events are processed by the event handler methods specified in this interface.

A message filter consisting of message fields (for example MessageId or Priority) may be specified so that the listening object only receives events concerning messages that include the same fields as those specified. If no fields are specified, events are triggered for all messages on the queue.

Parameters

listener	A reference to the subscribing object
queueName	A String containing the name of the queue from which the listener wishes to receive events

params *null*, or an MQeFields object containing message fields. A value of *null* means that the listener wishes to receive events for all messages on the queue. Specifying an MQeFields object containing message fields means that the listener is only interested in events concerning messages whose fields match those contained in the filter.

Return Values

none

Exceptions

MQeException	Except_QMgr_NotActive
	Except_QMgr_QdoesNotExist

Example

```
class MyMQeApplication implements MQeMessageListenerInterface
{
    ...
    MQeFields filter = new MQeFields(); /* search parameters */
    filter.putByte( MQe.Msg_Priority,(byte)3); /* only interested in */
                                           /* msgs of priority 3 */

    ...
    /* add listener */
    MyQM.addMessageListener( this, "MyQueue", filter );
    ...
    /* Message arrived event handler */
    public void messageArrived( MQeMessageEvent msgEvent )
    {
        ...
        /* is it the Queue we are interested in?? */
        if ( msgEvent.getQueueName().equals("MyQueue") )
        {
            ...
        }
        ...
    }
}
```

Related Functions

- **removeMessageListener**

MQeQueueManager browseMessages**Syntax**

```
public MQeEnumeration browseMessages( String qmgrName,
                                     String queueName,
                                     MQeFields filter,
                                     MQeAttribute attribute,
                                     boolean justUID ) throws Exception;
```

Description

This method returns an enumeration of the messages available on a specified queue. The messages are not deleted from the queue. The queue can belong to a local or remote queue manager.

A filter can be specified, consisting of message fields (for example MessageId or Priority). This causes only messages that have matching fields to be returned

Returning an enumeration of messages in their entirety can be expensive in terms of system resources, so if the **justUID** parameter is set to **true**, just the unique ids of the messages that match the filter are returned.

MQeQueueManager

The messages returned in the enumeration are still visible to other MQSeries Everyplace applications. Therefore, when performing subsequent operations on the messages contained in the enumeration, the application should be aware that it is possible for another application to have processed these messages in the time since the enumeration was returned. To lock the messages contained in the enumeration, therefore preventing other applications from processing them, use the **browseMessagesAndLock** method.

Parameters

qmgrname

A string containing the name of the queue manager that holds the queue to be browsed. If a value of *null* is used it is assumed that the local queue manager is to be used.

queueName

A string containing the name of the queue to browse

filter

null, or an **MQeFields** object containing the parameters with which to perform the browse.

attribute

An **MQeAttribute** object used to provide message-level security.

justUID

A boolean value denoting whether to return the all the fields in the messages, or just the unique id values.

Return values

An **MQeEnumeration** containing zero or more **MQeMsgObject** message objects.

Exceptions

MQeException

Except_QMgr_NotActive

Except_QMgr_InvalidQMGrName

Except_QMgr_QDoesNotExist

Various other exceptions

Example

```
class MyMQeApplication
{
    ...
    MQeEnumeration msgs = null;
    byte[] msgId = MQe.asciiToByte("240999");
    byte[] correlId = MQe.asciiToByte("240999/2");

    try
    {
        /* setup parameters object for matching */
        MQeFields filter = new MQeFields(); /* match against msgs */
        filter.putArrayOfByte( MQe.Msg_MsgID, msgId ); /* with this Msg Id */
        filter.putArrayOfByte( MQe.Msg_CorrelID, /* & this Correl Id */
                               correlId );

        /* look at available messages */
        msgs = qmgr.browseMessages( null, "MyQueue", filter, null, false );
        ...
        /* get this one and remove from queue */
        MQeMsgObject msgObj = qmgr.getMessage( null, "MyQueue",
```

```

        (MQeFields)msgs.nextElement(),
        null, 0 );
    }
    catch ( MQeException e )
    {
        ...
    }
    ...
}

```

Related Functions

- **browseMessagesAndLock**

MQQueueManager browseMessagesAndLock

Syntax

```

public MQeEnumeration browseMessagesAndLock( String qmgrName,
                                             String queueName,
                                             MQeFields filter,
                                             MQeAttribute attribute,
                                             long confirmId,
                                             boolean justUID ) throws Exception;

```

Description

This method returns an enumeration of the messages available on a specified queue. The messages are not deleted from the queue. The queue may belong a local or remote queue manager.

A filter can be specified, which consists of message fields (MessageId and Priority for example), so that only messages that have matching fields are returned.

Any messages that are returned by this operation are also locked on the queue. This means that these messages still exist on the queue, but they will not be visible to any subsequent operations, until they are unlocked.

A lock id is returned as part of the browse enumeration. The lock ID allows operations to be performed on locked messages so long as it is specified as part of the message filter that is passed into that operation.

Lock ids are unique, so every browse and lock operation generates a different id. The lock id applies to all the messages that are returned by the browse operation.

The operations that can be performed on a locked message are:

- **getMessage()**
- **deleteMessage()**
- **unlockMessage()**
- **waitForMessage()**

Returning an enumeration of messages in their entirety can be expensive in terms of system resources, so if the **justUID** parameter is set to true just the unique ids of the messages that match the filter are returned.

Specifying an **MQeAttribute** object allows browsing of messages that have message-level security defined with a matching attribute. Browsing queues containing messages with differing levels of message-level security may cause undefined results.

The **confirmID** is used in the event of an error whilst executing this command. The error could occur before the lock id is returned to the

MQeQueueManager

application and yet leave the messages in a locked state on the target queue. Passing the same confirm id used on this method to the `undo` method restores the messages to their previous state. It is recommended that a unique value be used for each browse and lock operation. A unique value can be generated using the `MQe.uniqueValue()` method.

Parameters

qmgrName	A string containing the name of the queue manager that holds the queue to be browsed. If a value of <i>null</i> is used it is assumed that the local queue manager is to be used.
queueName	A string containing the name of the queue to browse.
filter	<i>null</i> , or an MQeFields object containing the message fields with which to perform the browse.
attribute	An MQeAttribute object used to provide message-level security.
confirmId	A long value that is used in the event of a queue manager failure. The application should store the value used, and use it to reset the messages should a failure occur.
justUID	A boolean value denoting whether to return the entire message, or just its unique id.

Return Values

An **MQeEnumeration** containing zero or more **MQeMsgObject** message objects. The enumeration also contains the lock id, which can be accessed using the `getLockID()` method.

Exceptions

MQeException	Except_QMgr_NotActive Except_QMgr_InvalidQMgrName Except_QMgr_QDoesNotExist
---------------------	---

Various other exceptions

Example

```
class MyMQeApplication extends MQe
{
    ...
    MQeEnumeration msgs = null;
    byte[] msgId = asciiToByte("240999");
    byte[] correlId = asciiToByte("240999/2");

    try
    {
        /* setup parameters object for matching */
        MQeFields filter = new MQeFields(); /* match against msgs */
        filter.putArrayOfByte( MQe.Msg_MsgId, msgId ); /* with this Msg Id */
        filter.putArrayOfByte( MQe.Msg_CorrelId, /* & this Correl Id */
                               correlId );

        /* look at available messages */
        msgs = qmgr.browseMessagesAndLock( null, "MyQueue", filter, null, 0,
                                           false );
        long lockId = msgs.getLockId(); /* get Lock Id */

        filter.putLong( MQe.Msg_LockID, lockId ); /* Add lock Id */

        /* get the first locked message from queue */
    }
}
```

MQeQueueManager

```
MQeMsgObject msgObj = qmgr.getMessage( null, "MyQueue", filter, null, 0 );
}
catch ( MQException e )
{
    ...
}
...
}
```

Related Functions

- **getMessage**
- **waitForMessage**
- **browseMessagesAndLock**
- **unlockMessage**
- **deleteMessage**
- **undo**

MQeQueueManager checkActive

Syntax

```
public boolean checkActive()
```

Description

This method allows an application to determine whether or not the queue manager is active.

Parameters

none

Return Values

A boolean value denoting whether or not the queue manager is active

Exceptions

none

Example

```
class MyMQeApplication
{
    ...
    qmgr = new MQeQueueManager( startupParams );
    if ( qmgr.checkActive() ) /* verify that QMgr is active */
    {
        ...
    }
    else
        throw new Exception( "Queue Manager not active" );
}
```

MQeQueueManager close

Syntax

```
public void close() throws MQException
```

Description .

This method closes down the queue manager. It should be called by MQe applications when they have finished using the queue manager.

Parameters

none

Return Values

none

MQeQueueManager

Exceptions

MQeException

Except_QMgr_NotActive

Example

```
class MyMQeApplication
{
    ...
    try
    {
        qmgr.putMessage( null, "MyQueue", msgObj, null, 0 );
    }
    catch ( MQeException e )
    {
        ...
    }
    ...
    qmgr.close(); /* close QMgr */
}
```

Related Functions

- activate

MQeQueueManager confirmGetMessage

Syntax

```
public void confirmGetMessage( String qmgrName,
                               String queueName,
                               MQeFields filter ) throws Exception
```

Description

This method confirms the successful receipt of a message that was retrieved from a queue by a previous `getMessage` operation. The message remains locked on the target queue until the confirm flow is received.

Parameters

- | | |
|------------------|---|
| queueName | A string containing the name of the queue on which the message is held. |
| qmgrName | A string containing the name of the queue manager that holds the queue. If a value of <i>null</i> is used it is assumed that the local queue manager is to be used. |
| filter | An MQeFields object containing a message filter. The filter must contain the message's unique id for the operation to be successful. |

Return Values

none

Exceptions

MQeException

Except_NotFound

Note: This exception is thrown when attempting to confirm a message that has already been confirmed. If an application has reissued a confirm get message request then this exception can be treated as a successful return code.

Various other exceptions

Example

```

class MyMQeApplication
{
    ...
    /* generate a unique confirmId for this operation */
    long confirmId = MQe.uniqueValue();
    /* get next available msg - msg still locked on target queue */
    MQeMsgObject msg = qmgr.getMessage( "RemoteQMGr", "RemoteQueue", null,
    null, confirmId );
    /* confirm the successful Get */
    qmgr.confirmGetMessage( "RemoteQMGr", "RemoteQueue",
    msg.getMsgUIDFields() );
    ...
}

```

Related Functions

- `getMessage`

MQeQueueManager confirmPutMessage**Syntax**

```

public void confirmPutMessage( String qmgrName,
                             String queueName,
                             MQeFields filter ) throws Exception

```

Description

This method performs the confirmation of a previously successful `putMessage` operation.

Parameters

queueName	A string containing the name of the queue on which the message is held.
qmgrName	A string containing the name of the queue manager that holds the queue. If a value of <i>null</i> is used it is assumed that the local queue manager is to be used.
filter	An MQeFields object containing a message filter. The filter must contain the message's unique id for the operation to be successful.

Return Values

none

Exceptions

MQeException	<code>Except_NotFound</code>
---------------------	------------------------------

Note: This exception is thrown when attempting to confirm a message that has already been confirmed. If an application has reissued a confirm put message request then this exception can be treated as a successful return code.

Various other exceptions

Example

```

class MyMQeApplication
{
    ...
    /* generate a unique confirmId for this operation */
    long confirmId = MQe.uniqueValue();

```

MQeQueueManager

```
qmgr.putMessage( "RemoteQMgr", "RemoteQueue", msg, null,
                confirmId );
/* confirm the put */
qmgr.confirmPutMessage( "RemoteQMgr", "RemoteQueue",
                        msg.getMsgUIDFields() );
...
}
```

Related Functions

- `putMessage`

MQeQueueManager deleteMessage

Syntax

```
public void deleteMessage( String qmgrName,
                          String queueName,
                          MQeFields filter ) throws MQeException
```

Description

This method deletes a message from a queue. It does not return the message to the application that called it.

Only one message can be deleted per operation and the unique id(timestamp and origin queue manager name) of the message must always be supplied.

The queue may belong to a local or remote MQSeries Everyplace queue manager.

Messages that have been locked by a previous operation (browse for example) can be deleted by included a valid lock id in the message filter.

If the message is not available, an exception is thrown.

Parameters

queueName	A String containing the name of the queue on which the message is held.
qmgrName	A String containing the name of the queue manager that holds the queue. If a value of <i>null</i> is used it is assumed that the queue manager is local.
filter	An MQeFields object containing a message filter. The filter must contain the unique id of the message for the operation to be successful.

Return Values

none

Exceptions

MQeException	Except_QMgr_InvalidQName Except_QMgr_NotActive Except_QMgr_QDoesNotExist Except_QMgr_WrongType Except_NotFound Except_NotAllowed
---------------------	---

Various other exceptions

Examples

```

class MyMQeApplication
{
    ...
    MQeEnumeration msgEnum;
    ...
    MQeFields filter = new MQeFields();
    filter.putArrayOfByte( MQe.Msg_MsgID, new byte[]{ 1,2,3,4 } );
    /* return all messages with a Message Id of 1234 */
    msgEnum = qmgr.browseMessages( null, "MyQueue", filter, null, false );
    /* delete all message with a Message Id of 1234 */
    while( msgEnum.hasMoreElements() )
        qmgr.deleteMessage( null, "MyQueue",
            (MQeMsgObject)msgEnum.nextElement() );
    ...
}

```

Related Functions

- **waitForMessage**
- **browseMessages**
- **browseMessagesAndLock**
- **putMessage**
- **getMessage**

MQeQueueManager getMessage**Syntax**

```

public MQeMsgObject getMessage( String qmgrName,
                                String queueName,
                                MQeFields filter,
                                MQeAttribute attribute,
                                long confirmId ) throws MQeException;

```

Description

This method returns an available message from the specified queue and the message is removed from the queue. The queue can belong to a local or a remote MQSeries Everyplace queue manager.

If no message filter is specified, the first available message on the queue is returned. If a message filter is specified, the first available message that matches the filter is returned.

Messages that have been locked by a previous browse operation can be retrieved by including, in the message filter, the lock id that was used to lock the message.

If no message is available, an exception is thrown.

The use of assured message delivery is dependent upon the value of the **confirmId** parameter. Passing a nonzero value returns the message as normal, but the message is locked and is not removed from the target queue until a subsequent confirm is received. A confirm can be issued using the **confirmGetMessage()** method. Passing a value of zero returns the message and removes it from the target queue, however, the message delivery is not assured.

The **confirmId** parameter is also used in the event of an error when executing this command. A failure could occur before the message is returned to the application and yet leave the message in a locked state on the target queue. Passing the same confirm id used for the get operation to the undo method restores the message to its previous state. It is

MQeQueueManager

recommended that a unique value be used for each get operation. A unique value can be generated using the **MQe.uniqueValue()** method.

Parameters

queueName	A string containing the name of the queue from which to obtain a message.
qmgrName	A string containing the name of the queue manager that holds the queue. If a value of <i>null</i> is used it is assumed that the queue manager is local.
filter	<i>null</i> , or an MQeFields object containing a message filter.
attribute	An MQeAttribute object used to provide message-level security. The attribute supplied must match any attribute attached to the message returned by this method. Failure to do this may result in message loss.
confirmId	A long value denoting whether or not to use guaranteed message delivery. A nonzero value does not remove the message from the target queue, this occurs on a subsequent confirm flow. A value of zero will remove the message from the target queue.

Return values

An **MQeMsgObject** containing the message obtained from the specified queue

Exceptions

MQeException	Except_QMgr_NotActive Except_QMgr_InvalidQName Except_QMgr_QDoesNotExist Except_QMgr_WrongQType Except_Q_NoMatchingMsg Except_NotFound
---------------------	---

Various other exceptions

Examples

Example 1—Simple get, no message filter

```
class MyMQeApplication
{
    ...
    try
    {
        /* get 1st available message on the queue */
        MQeMsgObject myMsgObject = qmgr.getMessage( null, "MyQueue", null, null,
                                                    0 );
    }
    catch ( MQeException e )
    {
        ...
    }
    ...
}
```

Example 2—Browse and get

```

class MyMQeApplication
{
    ...
    /* Lock all msgs on this queue */
    MQeEnumeration msgEnum = qmgr.browseMessagesAndLock( null, "MyQueue",
                                                         null, null, 0, false );

    long lockId = msgEnum.getLockId(); /* get the Lock Id */
    MQeFields filter = new MQeFields(); /* create a msg filter */
    filter.putLong( MQe.Msg_LockID, lockId ); /* add lock Id */
    /* get the 1st locked message on the queue */
    MQeMsgObject msgObj = qmgr.getMessage( null, "MyQueue", filter, null, 0 );
    ...
}

```

Example 3—get with assured message delivery

```

class MyMQeApplication
{
    ...
    /* generate a unique confirmId for this operation */
    long confirmId = MQe.uniqueValue();
    /* get next available msg - msg remains locked on the target queue */
    MQeMsgObject msg = qmgr.getMessage( "RemoteQMgr", "RemoteQueue", null,
                                       null, confirmId );

    /* confirm the successful Get */
    qmgr.confirmGetMessage( "RemoteQMgr", "RemoteQueue",
                           msg.getMsgUIDFields() );

    ...
}

```

Related functions

- **waitForMessage**
- **browseMessages**
- **browseMessagesAndLock**
- **putMessage**
- **deleteMessage**
- **confirmGetMessage**
- **undo**

MQeQueueManager getName

Syntax

```
public String getName();
```

Description

This method returns the name of this queue manager.

Note: It is strongly recommended that all queue manager names are unique within an MQSeries Everyplace network.

Parameters

none

Return values

A string containing the name of the queue manager.

Exceptions

none

Example

MQeQueueManager

```
class MyMQeApplication
{
    ...
    String qmgrName = qmgr.getName();
    ...
}
```

MQeQueueManager getReference

Syntax

```
public static MQeQueueManager getReference( String qmgrName) throws MQeException
```

Description

This method is used to obtain an object reference to an instantiated queue manager.

Parameters

qmgrName A String containing the name of an queue manager.

Return values

An MQeQueueManager object.

Exceptions

MQeException Except_QMgr_InvalidQMgrName

Example

```
class MyMQeApplication
{
    ...
    MQeQueueManager qmgr = null;
    ...
    /* Obtain a reference to "MyQMgr" Queue Manager */
    qmgr = MQeQueueManager.getReference( "MyQMgr" );
    /* Put a message */
    qmgr.putMessage( null, "DestQ", Msg, null, 0 );
    ...
}
```

MQeQueueManager putMessage

Syntax

```
public void putMessage( String qmgrName,
                       String queueName,
                       MQeMsgObject msg,
                       MQeAttribute attribute,
                       long confirmId ) throws Exception;
```

Description

This method places the specified message onto the specified queue. This queue may belong to a local or a remote queue manager.

Puts to remote queues can occur immediately, or at some later time depending upon how the remote queue is defined on the local queue manager.

If a remote queue is defined as synchronous, the transmission of the message over the network occurs immediately.

If a remote queue is defined as asynchronous, the message is stored within the local queue manager. The message remains there until the queue manager rules decide that it is time to transmit any pending messages or the queue manager is triggered through the **triggerTransmission()** method.

If the local queue manager does not hold a definition of the remote queue then it attempts to contact the queue synchronously.

The assured delivery of the message is dependent on the value of the **confirmId** parameter. Passing a nonzero value transmits the message as normal, but the message is locked on the target queue until a subsequent confirm is received. Passing a value of zero transmits the message without the need for a subsequent confirm, however the delivery of the message is not assured.

The **confirmID** is also used in the event of an error during the execution of this command. Passing the same confirm id used for the put operation to the undo method removes the unconfirmed message from the target queue. It is recommended that a unique value be used for each put operation. A unique value can be generated using the **MQe.uniqueValue()** method.

A message can be protected using message-level security (see *MQSeries Everyplace Programming Guide* for information on MQSeries Everyplace security). The security is defined by providing an **MQeAttribute** object, or one of its descendants. The attribute can be attached to the message prior to any put message request, or the attribute parameter can be used to specify the message-level security to be used.

If the attribute parameter is not *null*, the value overrides any attribute attached to the message prior to the put message request. If the attribute parameter is *null*, it has no effect on the sending of the message.

Parameters

queueName	A string containing the name of the queue on which the message should be placed.
qmgrName	A string containing the name of the remote queue manager to which the specified queue belongs. If a value of <i>null</i> is used it is assumed that the queue manager is local.
msg	An MQeMsgObject containing the message.
attribute	An MQeAttribute object or a descendant, or <i>null</i> . If <i>null</i> , then this parameter has no effect on the sending of the message. The attribute specified here over-rides any attribute that has been previously associated with the message, and MQeFields data in the message using previous calls to the MQeFields.setAttribute() or MQeMsgObject.setAttribute() methods. You may pass an MQeMAttribute or MQeMTrustAttribute to perform message-level security operations.
confirmId	A long value denoting whether or not to use assured message delivery. A nonzero value locks the message on the target queue, it is not made visible until a subsequent confirm flow. A value of zero transmits the message without the need for a subsequent confirm.

Return values

none.

MQeQueueManager

Exceptions

MQeException	Except_QMgr_InvalidQName
	Except_QMgr_NotActive
	Except_QMgr_QDoesNotExist
	Except_Duplicate

Various other exceptions

Example

Example 1—simple put

```
class MyMQeApplication
{
    ...
    try
    {
        qmgr.putMessage( null, "MyQueue", msgObj, /* simple put */
                        null, 0 );
    }
    catch ( MQeException e )
    {
        ...
    }
    ...
}
```

Example 2—put with assured message delivery

```
class MyMQeApplication
{
    ...
    /* generate a unique confirmId for this operation */
    long confirmId = MQe.uniqueValue();
    qmgr.putMessage( "RemoteQMgr", "RemoteQueue", msg, null, confirmId );
    /* confirm the put */
    qmgr.confirmPutMessage( "RemoteQMgr", "RemoteQueue",
                           msg.getMsgUIDFields() );
    ...
}
```

Related functions

- **getMessage**
- **waitForMessage**
- **confirmPutMessage**
- **undo**

MQeQueueManager removeMessageListener

Syntax

```
public void removeMessageListener( MQeMessageListenerInterface listener,
                                   String queueName,
                                   MQeFields filter ) throws MQeException
```

Description

This method removes an objects subscription to MQSeries Everyplace message events generated by the queue specified in **queueName**. It is only possible to have listeners on local queues.

Note: The listening object must implement the **MQeMessageListenerInterface**.

MQeQueueManager

If an optional message filter is specified, the object's subscription is only removed for events concerning messages that include the same fields as those specified in the filter. If the filter is *null*, the object's subscription for events concerning all messages is removed.

Parameters

listener	A reference to the subscribing object.
queueName	A String containing the name of the queue from which the listener wishes to receive events.
filter	<i>null</i> or a MQeFields object containing a message filter.

Return values

none

Exceptions

MQeException	Except_QMgr_NotActive Except_QMgr_InvalidQName Except_QMgr_QDoesNotExist
---------------------	--

Example

```
class MyMQeApplication implements MQeMessageListenerInterface
{
    ...
    /* remove the 'all messages' listener for this queue */
    qmgr.removeMessageListener( this, "MY.QUEUE", null );
    ...
}
```

Related Functions

addMessageListener

MQeQueueManager triggerTransmission

Syntax

```
public void triggerTransmission() throws Exception
```

Description

This method causes an attempt to transmit any pending messages.

Pending messages are messages awaiting transmission to remote queue managers. Typically, the transmission of pending messages would be handled by the queue manager rules, but this method allows transmission of pending messages at a time convenient to the application.

In addition, this method triggers any home server queues that are defined. These queues attempt to collect messages from their home servers.

This method overrides the operation of the `MQeQueueManagerRule.triggerTransmission()` rule, however, it does call the `MQeQueueManagerRule.transmi()` rule.

Parameters

none

Return values

none

Exceptions

MQeException	Except_BadRequest
---------------------	-------------------

MQeQueueManager

Except_QMgr_NotActive

Except_QMgr_QDoesNotExist

Various other exceptions

Examples

```
class MyMQeApplication
{
    ...
    try
    {
        if ( timeToTransmit() ) /* application decides it's time to */
            qmgr.triggerTransmission(); /* transmit */
    }
    catch ( MQeException e )
    {
        if ( e.Code() != Except_QMgr_Busy )
            throw e;
    }
    ...
}
```

MQeQueueManager undo

Syntax

```
public void undo( String qmgrName,
                 String queueName,
                 long confirmId ) throws Exception
```

Description

This method is intended to be used in the event of an error during a **put**, **get**, or **browseAndLock** command. It is possible that the error could leave messages in an unconfirmed or locked state on the target queue. This method resets the message to the state (either locked or unlocked) that it was in prior to the failed operation, or in the case of an unconfirmed put operation, the message is deleted.

To reset the message, it is necessary to supply the **confirmId** that was used in the failed operation. It is recommended that confirm ids are unique for each message. A unique value can be generated using the **MQe.uniqueValue()** method.

Parameters

qmgrName	A string containing the name of the queue manager that holds the queue. If a value of <i>null</i> is used, it is assumed that the local queue manager is the queue manager to be used.
queueName	A string containing the name of the queue that holds the locked message
confirmId	A long value that is the same as the confirm id used on the failed operation.

Return values

none.

Exceptions

MQeException	Except_QMgr_NotActive
	Except_QMgr_InvalidQName

Except_QMgr_QDoesNotExist

Except_Q_NoMatchingMsg

Except_NotAllowed

Various other exceptions

Examples

```
class MyMQeApplication
{
    ...
    /* generate a unique confirmId for this operation */
    long confirmId = MQe.uniqueValue();
    try
    {
        qmgr.putMessage( "RemoteQMgr", "RemoteQueue", msg, null, confirmId );
        qmgr.confirmPutMessage( "RemoteQMgr", "RemoteQueue",
                               msg.getMsgUIDFields() );
    }
    catch ( Exception e )
    {
        /* Give the remote Queue Manager time to recover from error */
        Thread.sleep( 30000 );
        /* Remote Queue Manager failure - undo the put message */
        qmgr.undo("RemoteQMgr", "RemoteQueue", confirmId );
    }
    ...
}
```

Related functions

- **browseMessagesAndLock**
- **getMessage**
- **putMessage**

MQeQueueManager unlockMessage

Syntax

```
public void unlockMessage( String qmgrName,
                          String queueName,
                          MQeFields filter ) throws Exception
```

Description

This method unlocks a message that has been previously locked. This makes it visible once again to all applications. Only one message can be unlocked at a time and both the unique id (timestamp and origin queue manager name), and lock id of the message must be supplied.

The queue may belong to a local or remote queue manager.

If the message is not available, an exception is thrown.

This method would typically be used in conjunction with the `browseMessagesAndLock()` method.

Parameters

qmgrName	A string containing the name of the queue manager that holds the queue. If a value of <i>'null'</i> is used, it is assumed that the local queue manager is to be used.
queueName	A string containing the name of the queue that holds the locked message.
filter	An MQeFields object containing a message filter. This

MQeQueueManager

must contain both the message unique id and lock id for the operation to be successful.

Return values

none

Exceptions

MQeException	Except_QMgr_NotActive
	Except_QMgr_InvalidQName
	Except_QMgr_QDoesNotExist
	Except_Q_NoMatchingMsg
	Except_NotAllowed

Various other exceptions

Example

```
class MyMQeApplication
{
    ...
    MQeEnumeration msgEnum;
    ...
    /* lock all msgs on queue */
    msgEnum = qmgr.browseMessagesAndLock( null, "MyQueue", null, null, 0,
                                         false );
    long lockID = msgEnum.getLockId(); /* get lockID */
    while( msgEnum.hasMoreElements() )
    {
        MQeFields msgFields = (MQeFields)msgEnum.nextElement();
        String msgID = byteToAscii( msgFields.getArrayOfByte( MQe.Msg_MsgID ) );
        /* Unlock all messages with an ID of 1234 */
        if ( msgID.equals("1234") )
        {
            msgFields.putLong( MQe.Msg_LockID, lockID );
            qmgr.unlockMessage( null, "MyQueue", msgFields );
        }
    }
    ...
}
```

Related Functions

- **browseMessageAndLock**

MQeQueueManager waitForMessage

Syntax

```
public MQeMsgObject waitForMessage( String qmgrName,
                                    String queueName,
                                    MQeFields filter,
                                    MQeAttribute attribute,
                                    long confirmId,
                                    int milliseconds ) throws MQeException;
```

Description

This method works in an identical manner to **getMessage**. However, if no message is available, the method waits for the period of time specified by **milliseconds**. If no message is available at the end of this period, an Exception is thrown.

Parameters

qmgrName	A string containing the name of the queue manager that holds the queue. If a value of <i>null</i> is used, it is assumed that the local queue manager is to be used.
queueName	a string containing the name of the MQSeries Everyplace queue from which to obtain a message.
filter	<i>null</i> , or an MQeFields object containing a message filter.
attribute	An MQeAttribute object used to provide message-level security.
confirmId	A long value denoting whether or not to use assured message delivery. A nonzero value does not remove the message from the target queue, this occurs on a subsequent confirm flow. A value of zero removes the message from the target queue.
milliseconds	The period of time (specified in milliseconds) for which to wait for a message to become available.

Return values

An **MQeMsgObject** containing the message obtained from the specified queue.

Exceptions

MQeException	Except_QMgr_NotActive Except_QMgr_InvalidQName Except_QMgr_QDoesNotExist Except_Q_NoMatchingMsg Except_Q_NoMsgAvailable
---------------------	---

Various other exceptions

Examples

```
class MyMQeApplication extends MQe
{
    ...
    String MsgId = "260399";
    String CorrelId = "260399/2";
    ...
    /* set up a parameters object to match with */
    /* only interested in msgs*/
    MQeFields filter = new MQeFields();
    /* with this message Id*/
    filter.putArrayOfByte( MQe.Msg_MsgID, asciiToByte( MsgId ) );
    /* & this correlation Id */
    filter.putArrayOfByte ( MQe.Msg_CorrelID, asciiToByte( CorrelId ) );
    ...
    /* wait 10 seconds for a msg to arrive */
    MQeMsgObject msgObj = qmgr.waitForMessage( null, "MyQueue", filter,
        null, 0, 10000 );
    ...
}
```

Related functions

- **getMessage**

MQeQueueManagerConfigure

This class is used to configure a queue manager. It is used to create and delete queue managers and their default queues.

Package `com.ibm.mqe`

Constructor summary

Constructor	Purpose
<code>MQeQueueManagerConfigure</code>	Instantiates the <code>QueueManagerConfigure</code> object

Method summary

Method	Purpose
<code>activate</code>	Activates the configuration object
<code>close</code>	Closes the configuration object
<code>defineDefaultAdminQueue</code>	Defines a standard default <i>administration queue</i> in the registry
<code>defineDefaultAdminReplyQueue</code>	Defines a standard default <i>administration reply queue</i> in the registry
<code>defineDefaultDeadLetterQueue</code>	Defines a standard default <i>dead letter queue</i> in the registry
<code>defineDefaultSystemQueue</code>	Defines a standard default local queue in the registry.
<code>defineQueueManager</code>	Defines a standard queue manager in the registry.
<code>deleteAdminQueueDefinition</code>	Deletes the <i>administration queue</i> from the registry.
<code>deleteAdminReplyQueueDefinition</code>	Deletes the <i>administration reply queue</i> from the registry.
<code>deleteDeadLetterQueueDefinition</code>	Deletes the <i>dead letter queue</i> from the registry.
<code>deleteQueueManagerDefinition</code>	Deletes the queue manager from the registry.
<code>deleteStandardQMDefinitions</code>	Deletes the queue manager and the standard queues from the registry.
<code>deleteSystemQueueDefinition</code>	Deletes the standard default local queue from the registry.
<code>queueManagerExists</code>	Checks whether the queue manager exists in the registry.
<code>setChannelTimeout</code>	Sets the Channel time-out value for the queue manager.
<code>setChnlAttributeRuleName</code>	Sets the name of the Channel Attribute Rule for the queue manager.
<code>setDescription</code>	Sets the description of the queue manager.

MQeQueueManagerConfigure

Syntax

- `public MQeQueueManagerConfigure()`
- `public MQeQueueManagerConfigure(MqeFields startupParameters) throws Exception`

MQeQueueManagerConfigure

```
3. public MQeQueueManagerConfigure( MqeFields startupParameters,  
                                   String qStore ) throws Exception
```

Description

The constructors instantiate the queue manager configuration object. There are three forms of the constructor:

1. This form is designed for dynamic loading and must be followed by a call to **activate()**
2. This form can be used to only for the deletion of a queue manager.
3. This form can be used for the creation or deletion of a queue manager

Parameters

startupParameters

An **MQeFields** object containing the initialization parameters for the queue manager. These are described in **MQeQueueManager startupParameters**.

qStore

The location where the standard default queues are stored. This must be specified if a queue manager is being created. If a queue manager is being deleted this parameter can be *null*.

Return Values

none

Exceptions

Exception - thrown if there is a problem initializing the queue manager configure object

Example

```
MQeQueueManagerConfigure qmConfig1;  
qmConfig1 = new MQeQueueManagerConfigure();  
  
try  
{  
    MQeQueueManagerConfigure qmConfig2;  
    MQeFields parms = new MQeFields();  
    // initialize the parameters  
    ...  
    qmConfig2 = new MQeQueueManagerConfigure( parms );  
}  
catch (Exception e)  
{ ... }
```

```
try  
{  
    MQeQueueManagerConfigure qmConfig3;  
    MQeFields parms = new MQeFields();  
    // initialize the parameters  
    ...  
    qmConfig3 = new MQeQueueManagerConfigure( parms, "qmName" + File.separator +  
                                              "Queues" + File.separator );  
}  
catch (Exception e)  
{ ... }
```

MQeQueueManagerConfigure activate

Syntax

```
public void activate( MqeFields startupParameters, String qStore ) throws Exception
```

MQeQueueManagerConfigure

Description

This method initializes the object ready to configure a queue manager.

Parameters

startupParameters

An **MQeFields** object containing the initialization parameters for the queue manager. These are described in **MQeQueueManager startupParameters**.

qStore

The location where the standard default queues are stored. This must be specified if a queue manager is being created. If a queue manager is being deleted, this parameter can be *null*.

Return Values

none

Exceptions

Exception - is thrown if there is a problem initializing the object.

Example

```
try
{
    MQeQueueManagerConfigure qmConfig;
    MQeFields parms = new MQeFields();
    // initialize the parameters
    ...
    qmConfig = new MQeQueueManagerConfigure( );
    qmConfig.activate( parms, "qmName" + File.separator +
        "Queues" + File.separator );
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure close

Syntax

```
public void close()
```

Description

This method closes the configuration object. An attempt to use the object after it has been closed will result in an exception. The configuration object must be closed before the queue manager itself can be activated.

Parameters

none

Return Values

none

Exceptions

none

Example

```
try
{
    MQeQueueManagerConfigure qmConfig;
    MQeFields parms = new MQeFields();
    // initialize the parameters
    ...
    qmConfig = new MQeQueueManagerConfigure( parms, "qmName" + File.separator +
        "Queues" + File.separator );
    ...
}
```



```

    qmConfig.close();
}
catch (Exception e)
{ ... }

```

MQeQueueManagerConfigure defineDefaultAdminQueue

Syntax

```
public void defineDefaultAdminQueue( ) throws Exception
```

Description

This method defines a standard *administration queue* in the registry for the queue manager. The queue itself will be created the first time it is accessed from the running queue manager. An exception is thrown if the queue already exists.

Parameters

none

Return Values

none

Exceptions

MQeException

Is thrown if the MQeQueueManagerConfigure object has not been activated or if the queue already exists in the queue manager's registry.

Exception

Is thrown for other errors.

Example

```

try
{
    MQeQueueManagerConfigure qmConfig;
    MQeFields parms = new MQeFields();
    // initialize the parameters
    ...
    qmConfig = new MQeQueueManagerConfigure( parms, "qmName" + File.separator +
                                             "Queues" + File.separator );
    qmConfig.defineDefaultAdminQueue();
    ...
    qmConfig.close();
}
catch (Exception e)
{ ... }

```

MQeQueueManagerConfigure defineDefaultAdminReplyQueue

Syntax

```
public void defineDefaultAdminReplyQueue ( ) throws Exception
```

Description

This method defines a standard *administration reply queue* in the registry for the queue manager. The queue itself will be created the first time it is accessed from the running queue manager. An exception is thrown if the queue already exists.

Parameters

none

Return Values

none

MQeQueueManagerConfigure

Exceptions

MQeException

Is thrown if the MQeQueueManagerConfigure object has not been activated, or if the queue already exists in the queue manager's registry

Exception

Is thrown for other errors.

Example

```
try
{
    MQeQueueManagerConfigure qmConfig;
    MQeFields parms = new MQeFields();
    // initialize the parameters
    ...
    qmConfig = new MQeQueueManagerConfigure( parms, "qmName" + File.separator +
                                             "Queues" + File.separator );
    qmConfig.defineDefaultAdminReplyQueue();
    ...
    qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure defineDefaultDeadLetterQueue

Syntax

```
public void defineDefaultDeadLetterQueue ( ) throws Exception
```

Description

This method defines a standard *dead letter queue* in the registry for the queue manager. The queue itself will be created the first time it is accessed from the running queue manager. An exception is thrown if the queue already exists.

Parameters

none

Return Values

none

Exceptions

MQeException

Is thrown if the MQeQueueManagerConfigure object has not been activated, or if the queue already exists in the queue manager's registry

Exception

Is thrown for other errors.

Example

```
try
{
    MQeQueueManagerConfigure qmConfig;
    MQeFields parms = new MQeFields();
    // initialize the parameters
    ...
    qmConfig = new MQeQueueManagerConfigure( parms, "qmName" + File.separator +
                                             "Queues" + File.separator );
    qmConfig.defineDefaultDeadLetterQueue();
    qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure defineDefaultSystemQueue**Syntax**

```
public void defineDefaultSystemQueue( ) throws Exception
```

Description

This method defines a standard local queue, called `SYSTEM.DEFAULT.LOCAL.QUEUE`, in the registry for the queue manager. The queue itself will be created the first time it is accessed from the running queue manager. An exception is thrown if the queue already exists.

Parameters

none

Return Values

none

Exceptions**MQeException**

Is thrown if the `MQeQueueManagerConfigure` object has not been activated, or if the queue already exists in the queue manager's registry

Exception

Is thrown for other errors.

Example

```
try
{
    MQeQueueManagerConfigure qmConfig;
    MQeFields parms = new MQeFields();
    // initialize the parameters
    ...
    qmConfig = new MQeQueueManagerConfigure( parms, "qmName" + File.separator +
                                             "Queues" + File.separator );
    qmConfig.defineDefaultSystemQueue();
    ...
    qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure defineQueueManager**Syntax**

```
public void defineQueueManager( ) throws Exception
```

Description

This method creates a definition for the queue manager in the registry. This is required before the queue manager itself can be activated. An exception is thrown if the queue manager definition already exists.

Parameters

none

Return Values

none

Exceptions**MQeException**

Is thrown if the `MQeQueueManagerConfigure` object has not been activated, or if the queue manager definition already exists in the registry.

MQeQueueManagerConfigure

Exception Is thrown for other errors.

Example

```
try
{
    MQeQueueManagerConfigure qmConfig;
    MQeFields parms = new MQeFields();
    // initialize the parameters
    ...
    qmConfig = new MQeQueueManagerConfigure( parms, "qmName" + File.separator +
                                             "Queues" + File.separator );
    qmConfig.setDescription( "queue manager for " + qmName );
    qmConfig.defineQueueManager();
    qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure deleteAdminQueueDefinition

Syntax

```
public void deleteAdminQueueDefinition( ) throws Exception
```

Description

This method deletes the definition of the standard *administration queue* from the registry for the queue manager. No error is generated if the definition does not exist. The queue itself is not removed.

The queue cannot be accessed if it is not defined in the registry. The definition can be recreated with **defineDefaultAdminQueue()**.

Parameters

none

Return Values

none

Exceptions

MQeException Is thrown if the MQeQueueManagerConfigure object has not been activated, or if there is an error deleting the registry entry.

Example

```
try
{
    MQeQueueManagerConfigure qmConfig;
    MQeFields parms = new MQeFields();
    // initialize the parameters
    ...
    qmConfig = new MQeQueueManagerConfigure( parms, null );
    qmConfig.deleteAdminQueueDefinition();
    ...
    qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure deleteAdminReplyQueueDefinition

Syntax

```
public void deleteAdminReplyQueueDefinition ( ) throws Exception
```

Description

This method deletes the definition of the standard *administration reply queue* from the registry for the queue manager. No error is generated if the definition does not exist. The queue itself is not removed.

The queue cannot be accessed if it is not defined in the registry. The definition can be recreated with **defineDefaultAdminReplyQueue()**.

Parameters

none

Return Values

none

Exceptions**MQeException**

Is thrown if the MQeQueueManagerConfigure object has not been activated, or if there is an error deleting the registry entry.

Example

```
try
{
    MQeQueueManagerConfigure qmConfig;
    MQeFields parms = new MQeFields();
    // initialize the parameters
    ...
    qmConfig = new MQeQueueManagerConfigure( parms, null );
    qmConfig.deleteAdminReplyQueueDefinition();
    ...
    qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure deleteDeadLetterQueueDefinition

Syntax

```
public void deleteDeadLetterQueueDefinition ( ) throws Exception
```

Description

This deletes the definition of the standard *dead letter queue* from the registry for the queue manager. No error is generated if the definition does not exist. The queue itself is not removed.

The queue cannot be accessed if it is not defined in the registry. The definition can be recreated with **defineDefaultDeadLetterQueue()**.

Parameters

none

Return Values

none

Exceptions**MQeException**

Is thrown if the MQeQueueManagerConfigure object has not been activated, or if there is an error deleting the registry entry.

Example

MQeQueueManagerConfigure

```
try
{
    MQeQueueManagerConfigure qmConfig;
    MQeFields parms = new MQeFields();
    // initialize the parameters
    ...
    qmConfig = new MQeQueueManagerConfigure( parms, null );
    qmConfig.deleteDeadLetterQueueDefinition();
    ...
    qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure deleteQueueManagerDefinition

Syntax

```
public void deleteQueueManagerDefinition ( ) throws Exception
```

Description

This method deletes the definition of the queue manager from its registry. No error is generated if the definition does not exist.

The queue cannot be accessed if it is not defined in the registry. The definition can be recreated with **defineQueueManager()**.

Parameters

none

Return Values

none

Exceptions

MQeException

is thrown if the MQeQueueManagerConfigure object has not been activated, or if there is an error deleting the registry entry.

Example

```
try
{
    MQeQueueManagerConfigure qmConfig;
    MQeFields parms = new MQeFields();
    // initialize the parameters
    ...
    qmConfig = new MQeQueueManagerConfigure( parms, null );
    ...
    qmConfig.deleteQueueManagerDefinition();
    qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure deleteStandardQMDefinitions

Syntax

```
public void deleteStandardQMDefinitions ( ) throws Exception
```

Description

This deletes the definitions of the standard default queues and the queue manager itself from the registry. No error is generated if the definitions do not exist.

This method is provided for convenience, it is equivalent to:

```

deleteDeadLetterQueueDefinition();
deleteSystemQueueDefinition();
deleteAdminQueueDefinition();
deleteAdminReplyQueueDefinition();
deleteQueueManagerDefinition();

```

Parameters

none

Return Values

none

Exceptions**MQeException**

Is thrown if the MQeQueueManagerConfigure object has not been activated, or if there is an error deleting the registry entries.

Example

```

try
{
    MQeQueueManagerConfigure qmConfig;
    MQeFields parms = new MQeFields();
    // initialize the parameters
    ...
    qmConfig = new MQeQueueManagerConfigure( parms, null );
    qmConfig.deleteStandardQMDefinitions();
    qmConfig.close();
}
catch (Exception e)
{ ... }

```

MQeQueueManagerConfigure deleteSystemQueueDefinition**Syntax**

```
public void deleteSystemQueueDefinition ( ) throws Exception
```

Description

This deletes the definition of the default local queue, SYSTEM.DEFAULT.LOCAL.QUEUE, from the registry for the queue manager. No error is generated if the definition does not exist. The queue itself is not removed.

The queue cannot be accessed if it is not defined in the registry. The definition can be recreated with **defineDefaultSystemQueue()**

Parameters

none

Return Values

none

Exceptions**MQeException**

Is thrown if the MQeQueueManagerConfigure object has not been activated, or if there is an error deleting the registry entry.

Example

MQeQueueManagerConfigure

```
try
    MQeQueueManagerConfigure qmConfig;
    MQeFields parms = new MQeFields();
    // initialize the parameters
    ...
    qmConfig = new MQeQueueManagerConfigure( parms, null );
    qmConfig.deleteSystemQueueDefinition();
    ...
    qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure queueManagerExists

Syntax

```
public boolean queueManagerExists( ) throws Exception
```

Description

This method checks whether the queue manager definition exists in the registry.

Parameters

none

Return Values

true If the queue manager definition exists in the registry.
false If the queue manager definition does not exist in the registry.

Exceptions

MQeException Is thrown if the MQeQueueManagerConfigure object has not been activated, or if there is an error reading the registry.

Example

```
try
{
    MQeQueueManagerConfigure qmConfig;
    MQeFields parms = new MQeFields();
    // initialize the parameters
    ...
    qmConfig = new MQeQueueManagerConfigure( parms, null );
    if ( queueManagerExists() )
    {
        ...
    }
    qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure setChannelTimeout

Syntax

```
public void setChannelTimeout( long ChnlTimeout )
```

Description

This sets the channel time-out value for the queue manager.

MQeQueueManagerConfigure

This method must be called before `defineQueueManager()`, otherwise it is ignored.

Parameters

ChnlTimeout The Channel time-out value in milliseconds.

Return Values

none

Exceptions

none

Example

```
try
{
    MQeQueueManagerConfigure qmConfig;
    MQeFields parms = new MQeFields();
    // initialize the parameters
    ...
    qmConfig = new MQeQueueManagerConfigure( parms, "qmName" + File.separator +
                                           "Queues" + File.separator );
    qmConfig.setChannelTimeout( 3600 * 1000 );
    qmConfig.defineQueueManager();
    qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure setChnlAttributeRuleName

Syntax

```
public void setChnlAttributeRuleName( String ChnlAttrRuleName ) throws MQeException
```

Description

This method sets the name of the **Channel Attribute Rule** class for the queue manager.

This method must be called before `defineQueueManager()`, otherwise it is ignored.

Parameters

ChnlAttrRuleName

The name of the **Channel Attribute Rule** class.

Return Values

none

Exceptions

MQeException

Is thrown if the name is invalid.

Example

```
try
{
    MQeQueueManagerConfigure qmConfig;
    MQeFields parms = new MQeFields();
    // initialize the parameters
    ...
    qmConfig = new MQeQueueManagerConfigure( parms, "qmName" + File.separator +
                                           "Queues" + File.separator );
    qmConfig.setChnlAttributeRuleName( "Examples.Rules.AttributeRule" );
    qmConfig.defineQueueManager();
}
```

MQeQueueManagerConfigure

```
        qmConfig.close();
    }
    catch (Exception e)
    { ... }
```

MQeQueueManagerConfigure setDescription

Syntax

```
public void setDescription (String description )
```

Description

This method sets the description for the queue manager

This method must be called before **defineQueueManager()**, otherwise it is ignored.

Parameters

description The new description

Return Values

none

Exceptions

none

Example

```
try
{
    MQeQueueManagerConfigure qmConfig;
    MQeFields parms = new MQeFields();
    // initialize the parameters
    ...
    qmConfig = new MQeQueueManagerConfigure( parms, "qmName" + File.separator +
                                             "Queues" + File.separator );
    qmConfig.setDescription( "queue manager for " + qmName );
    qmConfig.defineQueueManager();
    qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerRule

This class contains methods that are invoked when the queue manager performs certain operations. The rules can affect the outcome of these operations. This class contains the default queue manager rules. Typically, these default rules would be overridden to provide appropriate behavior for a given MQe solution.

Package

com.ibm.mqe

Method summary

Method	Purpose
activateQueues	This rule decides whether to activate certain queues at queue manager startup time. The queues that can be activated are remote asynchronous queue definitions, <i>home server queues</i> , and <i>store and forward queues</i> .
addQueue	This rule is called when a new queue is added to the queue manager.
deleteMessage	This rule is called when a delete message operation occurs.
getMessage	This rule is called when a get message operation occurs.
getRetryCount	This rule returns the number of retry attempts for a failed network operation.
peerConnection	This rule is called when the queue manager's peer channel listener receives an incoming connection request.
putMessage	This rule is called when a put message operation occurs.
queueManagerActivate	This rule is called when the queue manager is activated.
queueManagerClose	This rule is called when the queue manager is closed.
removeQueue	This rule is called when a queue is to be removed from the queue manager.
transmit	This rule is called for each remote asynchronous queue definition when a transmission of pending messages is taking place. The rule allows transmission to be disabled on a per-queue basis.
triggerTransmission	This rule returns a boolean value which denotes whether or not to allow, at this time, the transmission of pending messages stored within remote asynchronous queue definitions
undo	This rule is called when an Undo operation occurs.

MQeQueueManagerRule activateQueues

Syntax

```
public boolean activateQueues()
```

Description

This rule determines whether to activate certain queues at queue manager startup time. The queues that can be activated are remote asynchronous queue definitions, *home server queues*, and *store and forward queues*.

Activating these queues means that an attempt is made to transmit any messages that they hold. Queues are normally not activated until an operation is performed on them. It can be useful to activate these queues

MQeQueueManagerRule

immediately on queue manager startup because they may have transmission timer threads, or other functions associated with them.

Parameters

none

Return values

A boolean value denoting whether to activate certain queues at queue manager startup time. The queue manager acts on the value returned

Exceptions

none

Examples

```
class exampleRules extends MQeQueueManagerRule
{
    ...
    /* cheap rate transmission period start and end times      */
    protected int cheapRatePeriodStart = 18; /* 18:00 hrs */
    protected int cheapRatePeriodEnd = 9; /* 09:00 hrs */

    public boolean activateQueues()
    {
        super.activateQueues();
        if ( timeToTransmit() ) /* if OK to transmit */
            return true; /* then activate queues */
        else /* otherwise*/
            return false; /* don't activate queues */
    }

    /* This method determines if the current time is inside the defined */
    /* cheap rate period of transmission */
    protected boolean timeToTransmit()
    {
        /* get current time */
        long currentTimeLong = System.currentTimeMillis();

        Date date = new Date( currentTimeLong );
        Calendar calendar = Calendar.getInstance();
        calendar.setTime( date );

        /* get hour */
        int hour = calendar.get( Calendar.HOUR_OF_DAY );

        if ( hour >= cheapRatePeriodStart || hour < cheapRatePeriodEnd )
            return true; /* cheap rate */
        else
            return false; /* not cheap rate */
    }
    ...
}
```

MQeQueueManagerRule addQueue

Syntax

```
public void addQueue( MQeQueue queue ) throws Exception
```

Description

This rule is called when a queue is added to the queue manager. The rule is called before the addition of the queue, and so the rule is able to reject the operation by throwing an exception.

This rule also decides whether to activate the queue immediately or to activate the queue when it is first used. The default behaviour of the rule

is to activate all home-server queues immediately. All other types of queue are activated when they are first used.

Parameters

queue An **MQeQueue** object that is being added to the queue manager.

Return values

A boolean value denoting whether the queue should be activated immediately, or when it is first used.

true Activate the queue immediately

false Activate the queue when first used

Exceptions

none

Examples

```
class exampleRules extends MQeQueueManagerRule
{
    ...
    /* Don't allow asynchronous queues to be added to this Queue Manager */
    public boolean addQueue( MQeQueue queue ) throws Exception
    {
        boolean result = super.addQueue( queue );
        int accessMode = queue.getAccessMode();
        if ( accessMode == MQeQueue.Queue_ASynchronous )
            throw new MQeException( Except_Rule, "No Asynch Queues" );
        return ( result );
    }
    ...
}
```

Related functions

removeQueue

MQeQueueManagerRule deleteMessage

Syntax

```
public void deleteMessage( String destQMgrName,
                          String destQName,
                          MQeFields filter ) throws Exception
```

Description

This rule is called when a delete message operation takes place. The rule is called before the operation takes place, and so the rule can stop the operation by throwing an exception.

Parameters

destQMgrName

A String containing the name of the queue manager that owns the queue on which this operation takes place. A value of *null* denotes that the local queue manager is to be used

destQName

A String containing the name of the queue on which this operation takes place

filter

This is the filter to be used for the delete message operation. It is an **MQeFields** object containing message fields (for example, priority and messageID)

MQeQueueManagerRule

Return values

none

Exceptions

none

Examples

```
class exampleRules extends MQeQueueManagerRule
{
    ...
    /* This rule blocks message deletes on 'TopSecretQueue' */
    public void deleteMessage( String destQMgr, String destQ, MQeFields filter )
    {
        super.deleteMessage( destQMgr, destQ, filter );
        if( destQMgr == null || destQMgr.equals( Owner.GetName() ) )
        {
            if ( destQ.equals( "TopSecretQueue" ) )
                throw new MQeException( Except_Rule, "Can't delete on this Queue" );
        }
    }
    ...
}
```

MQeQueueManagerRule getMessage

Syntax

```
public void getMessage( String destQMgrName,
                        String destQName,
                        MQeFields filter,
                        MQeAttribute attribute,
                        long confirmId ) throws Exception
```

Description

This rule is called when a get message operation takes place. The rule is called before the operation takes place, and so the rule can stop the operation by throwing an exception.

Parameters

destQMgrName

A String containing the name of the queue manager that owns the queue on which this operation takes place. A value of *null* denotes that the local queue manager is to be used

destQName

A String containing the name of the queue on which this operation takes place

filter

This is the filter to be used for the get message operation. It is an **MQeFields** object containing message fields (for example, priority and messageID)

attribute

An **MQe Attribute** object used to provide message-level security

confirmId

A long value denoting whether or not to use guaranteed message delivery. A nonzero value leaves the message locked on the target queue, it is not removed until a subsequent confirm flow. A value of zero removes the message from the target queue without the need for a subsequent confirm.

MQeQueueManagerRule

This value is also used in the event of a get message failure. The application should store the value used, and use it to reset the messages (via the **undo** command) matched by the Get to their previous state.

Return values

none

Exceptions

none

Example

```
class exampleRules extends MQeQueueManagerRule
{
    ...
    /* This rule only allows GETs from 'OutboundQueue', if a password is */
    /* supplied as part of the filter */
    public void getMessage( String destQMgr, String destQ, MQeFields filter,
                           MQeAttribute attr, long confirmId )
    {
        super.getMessage( destQMgr, destQ, filter, attr, confirmId );
        if ( destQMgr.equals( Owner.GetName() ) && destQ.equals( "OutboundQueue" ) )
        {
            if ( !(filter.Contains( "Password" ) )
                throw new MQeException( Except_Rule, "Password not supplied" );
            else
            {
                String pwd = filter.getAscii( "Password" );
                if ( !(pwd.equals( "1234" ) )
                    throw new MQeException( Except_Rule, "Incorrect password" );
            }
        }
    }
    ...
}
```

Related functions

putMessage

MQeQueueManagerRule getRetryCount

Syntax

```
public int getRetryCount()
```

Description

This rule returns the number of times to retry a network operation. The queue manager calls this rule when creating a new channel object. The value returned by this rule is passed to the channel, and it is used in the event of a network operation failure.

Parameters

none

Return values

An integer value that contains the number of times to retry a network operation. The queue manager acts on the value returned.

Exceptions

none

Examples

```
class exampleRules extends MQeQueueManagerRule
{
    ...
}
```

MQeQueueManagerRule

```
    public int getRetryCount()
    {
        return (2); /* retry a network operation twice */
    }
    ...
}
```

MQeQueueManagerRule peerConnection

Syntax

```
public void peerConnection( String qmgrName )
```

Description

This method is called when a queue manager's peer listener detects an incoming connection request from another MQSeries Everyplace queue manager. The connection must be made over an **MQePeerChannel**, or its descendant.

By throwing an exception the rule can block the connection attempt.

Parameters

qmgrName A String containing the name of the MQSeries Everyplace queue manager that is requesting a connection

Return values

none

Exceptions

none

Examples

```
class exampleRules extends MQeQueueManagerRule
{
    ...
    public void peerConnection( String qmgrName )
    {
        /* block any connection attempt from 'RogueQMgr' */
        if ( qmgrName.equals( "RogueQMgr" ) )
            throw new MQeException( Except_Rule, "Connection not allowed" );
    }
    ...
}
```

MQeQueueManagerRule putMessage

Syntax

```
public void putMessage( String destQMGrName,
                       String destQName,
                       MQeMsgObject msg,
                       MQeAttribute attribute,
                       long confirmId ) throws Exception
```

Description

This rule is called when a put message operation takes place. The rule is called before the operation takes place, so the rule can stop the operation by throwing an exception.

Parameters

destQMGrName A String containing the name of the queue manager that

MQeQueueManagerRule

owns the queue on which this operation takes place. A value of *null* denotes that the local queue manager is to be used

destQName	A String containing the name of the queue on which this operation takes place
msg	The message object that is being placed on the target queue
attribute	<i>null</i> , or an MQeAttribute object defining the authenticator, cryptor, and compressor to be associated with this message.
confirmId	A long value denoting whether or not to use assures message delivery. A nonzero value locks the message on the target queue, it is not made visible until a subsequent confirm flow. A value of zero transmits the message without the need for a subsequent confirm.

Also, this value can be used in the event of a put message failure. By passing this value to the undo command, the application can remove any messages that were left in an incomplete state by the failed put operation.

Return values

none

Exceptions

none

Examples

```
class exampleRules extends MQeQueueManagerRule
{
    ...
    /* Only allow msgs containing an ID field to be placed on the Queue */
    public void putMessage( String destQMgr, String destQ, MQeMsgObject msg,
        MQeAttribute attribute, long confirmId )
    {
        if ( !(msg.Contains( MQe.Msg_MsgId )) )
            throw new MQeException( Except_Rule, "Msg must contain an ID" );
    }
    ...
}
```

Related functions

getMessage

MQeQueueManagerRule queueManagerActivate

Syntax

```
MQeQueueManagerRule queueManagerActivate
```

Description

This rule is called when the queue manager is activated.

Parameters

none

Return values

none

Exceptions

none

MQeQueueManagerRule

Examples

```
class exampleRules extends MQeQueueManagerRule
{
    ...
    /* default interval between triggers is 60 minutes */
    protected int    triggerInterval = 360000;
    /* background thread reference */
    protected Thread th    = null;
    /* Called when the Queue manager is activated */
    public void queueManagerActivate( ) throws Exception
    {
        super.queueManagerActivate();
        /* background thread which triggers transmission */
        th = new Thread( this, "TriggerThread" );
        th.start();          /* start timer thread */
    }

    /* Called when a Queue manager Close is called */
    public void queueManagerClose( ) throws Exception
    {
        super.QueueManagerClose();
        th.stop();          /* stop background thread on QMgr close*/
    }

    /* Background thread run method */
    /* Triggers transmission every interval until thread is stopped */
    public void run()
    {
        try
        {
            while ( true )
            {
                Thread.sleep( triggerInterval );          /* sleep for specified interval */
                /* check if ok to transmit */
                if ( triggerTransmission( 0, null ) )
                    /* trigger transmission on QMgr (which is rule owner) */
                    ((MQeQueueManager)Owner).triggerTransmission();
            }
        } catch ( Exception e )
        {
            e.printStackTrace( System.err );
        }
    }
    ...
}
```

Related functions

`queueManagerClose`

MQeQueueManagerRule queueManagerClose

Syntax

```
public void queueManagerClose() throws Exception
```

Description

This rule is called when the queue manager is closing.

Parameters

none

Return values

none

Exceptions

none

Examples

```

class exampleRules extends MQQueueManagerRule
{
    ...
    /* default interval between triggers is 60 minutes */
    protected int    triggerInterval = 360000;
    /* background thread reference */
    protected Thread th    = null;
    /* Called when the Queue manager is activated */
    public void queueManagerActivate( ) throws Exception
    {
        super.queueManagerActivate();
        /* background thread which triggers transmission */
        th = new Thread( this, "TriggerThread" );
        th.start();          /* start timer thread */
    }

    /* Called when a Queue manager Close is called */
    public void queueManagerClose( ) throws Exception
    {
        super.queueManagerClose();
        th.stop();          /* stop background thread on QMgr close*/
    }

    /* Background thread run method */
    /* Triggers transmission every interval until thread is stopped */
    public void run()
    {
        try
        {
            while ( true )
            {
                Thread.sleep( triggerInterval );          /* sleep for specified interval */
                /* check if ok to transmit */
                if ( triggerTransmission( 0, null ) )
                /* trigger transmission on QMgr (which is rule owner) */
                    ((MQQueueManager)Owner).triggerTransmission();
            }
        } catch ( Exception e )
        {
            e.printStackTrace( System.err );
        }
    }
    ...
}

```

Related functions

queueManagerActivate

MQQueueManagerRule removeQueue

Syntax

```
public void removeQueue( MQQueue queue) throws Exception
```

Description

This rule is called when a queue is to be removed from the queue manager. The rule is called before the removal of the queue, so the rule is able to reject the operation by throwing an exception.

Parameters

queue An **MQQueue** object that is to be removed from the queue manager

Return values

none

MQeQueueManagerRule

Exceptions

none

Examples

```
class exampleRules extends MQeQueueManagerRule
{
    ...
    /* This rule prevents the removal of the Payroll Queue */
    public void removeQueue( MQeQueue queue ) throws Exception
    {
        if ( queue.getQueueName().equals( "PayrollQueue" ) )
            throw new MQeException( Except_Rule, "Can't delete this queue" );
    }
    ...
}
```

Related functions

addQueue

MQeQueueManagerRule transmit

Syntax

```
public boolean transmit( MQeQueue queue )
```

Description

When a queue manager attempts to send all its pending messages, this rule is called for each queue that contains messages awaiting transmission. This rule decides whether to allow the transmission of those messages for the supplied queue.

Parameters

queue A **MQeQueue** object that holds messages awaiting transmission.

Return values

A boolean value denoting whether to allow the transmission of the messages held on this queue. The queue manager acts on the value returned.

Exceptions

none

Examples

```
class exampleRules extends MQeQueueManagerRule
{
    ...
    /* cheap rate transmission period start and end times */
    protected int cheapRatePeriodStart = 18;            /* 18:00 hrs */
    protected int cheapRatePeriodEnd = 9;            /* 09:00 hrs */

    /* This rule allows queue transmission if current time is during the cheap rate transmission period
    /* If the current time is not during the cheap rate transmission period
    /* then transmission is only allowed if the queue is high priority
    public boolean transmit( MQeQueue queue )
    {
        if ( timeToTransmit() )
            return true;            /* cheap rate */
        else
            if ( queue.GetPriority() > 4 )
                return true;            /* high priority Q */
    }

    /* This method determines if the current time is inside the defined */
}
```

```

/* cheap rate period of transmission */
protected boolean timeToTransmit()
{
    /* get current time */
    long currentTimeLong = System.currentTimeMillis();

    Date date = new Date( currentTimeLong );
    Calendar calendar = Calendar.getInstance();
    calendar.setTime( date );

    /* get hour */
    int hour = calendar.get( Calendar.HOUR_OF_DAY );

    if ( hour >= cheapRatePeriodStart || hour < cheapRatePeriodEnd )
        return true; /* cheap rate */
    else
        return false; /* not cheap rate */
}
...
}

```

Related functions**triggerTransmission****MQeQueueManagerRule triggerTransmission****Syntax**

```
public boolean triggerTransmission( int noofMsgs, MQeFields msgFields )
```

Description

This method authorizes the transmission of pending messages stored on remote asynchronous queue definitions within the queue manager.

This rule is invoked by the queue manager in two situations ·

- When the queue manager is instructed to transmit all of its pending messages (using the **MQeQueueManager.triggerTransmission()** method)
- When a message is sent to a remote queue that is defined as asynchronous. The queue manager invokes this rule to see whether to transmit all pending messages

Parameters

noofMsgs	The number of messages awaiting transmission on remote asynchronous queues
msgFields	This parameter is <i>null</i> if this rule has been invoked because the queue manager has been instructed to transmit all of its pending messages (using the MQeQueueManager.triggerTransmission() method.

If this rule has been invoked because of the arrival of a message on a remote asynchronous queue definition then this parameter is an **MQeFields** object containing certain fields from the newly-arrived message.

The fields present in the parameter are: ·

- Message UID (Origin queue manager + Timestamp)
- Message ID (if present in the original message)
- Correlation ID (if present in the original message)
- Priority (if present in the original message)

MQeQueueManagerRule

Return values

A boolean value denoting whether the rule allows transmission of pending messages at this time. The queue manager acts upon the value returned.

Exceptions

none

Examples

```
class exampleRules extends MQeQueueManagerRule
{
    ...
    /* default interval between triggers is 60 minutes */
    protected int    triggerInterval = 360000;
    /* background thread reference */
    protected Thread th    = null;
    /* Called when the Queue manager is activated */
    public void queueManagerActivate( ) throws Exception
    {
        super.queueManagerActivate();
        /* background thread which triggers transmission */
        th = new Thread( this, "TriggerThread" );
        th.start();          /* start timer thread */
    }

    /* Called when a Queue manager Close is called */
    public void queueManagerClose( ) throws Exception
    {
        super.queueManagerClose();
        th.stop();          /* stop background thread on QMgr close*/
    }

    /* Background thread run method */
    /* Triggers transmission every interval until thread is stopped */
    public void run()
    {
        try
        {
            while ( true )
            {
                /* sleep for specified interval */
                Thread.sleep( triggerInterval );
                /* check if ok to transmit */
                if ( triggerTransmission( 0, null ) )
                /* trigger transmission on QMgr (which is rule owner) */
                    ((MQeQueueManager)Owner).triggerTransmission();
            }
        } catch ( Exception e )
        {
            e.printStackTrace( System.err );
        }
    }

    /* Decides if transmission of messages is allowed */
    public boolean triggerTransmission( int noOfMsgs, MQeFields msgFields )
    {
        return true; /* always allow transmission */
    }
    ...
}
```

Related functions

transmit

MQeQueueManagerRule undo

Syntax

```
public void undo(String destQMgrName,
                 String destQName,
                 long confirmId ) throws Exception
```

Description

This rule is called when an undo message operation takes place. The rule is called before the operation takes place, so the rule can stop the operation by throwing an exception.

Parameters

destQMgrName

A String containing the name of the queue manager that owns the queue on which this operation takes place. A value of *null* denotes that the local queue manager is to be used

destQName

A String containing the name of the queue on which this operation takes place

confirmId

A long value that was the confirm Id used by the operation which is being undone. All messages matching this confirm Id are restored to their previous state

Return values

none

Exceptions

none

Examples

```
class exampleRules extends MQeQueueManagerRule
{
    ...
    public void undo( String destQMgr, String destQ, long confirmId )
    {
        /* log the undo event */
        log( MQe_Log_Warning, Event_QueueManager_Undo,
            destQMgr + "+" + destQ );
    }
    ...
}
```

MQeQueueRule

Queue rules control the behaviour of MQSeries Everyplace queues. The queue rules are activated by the queue when it is itself activated. During the operation of the queue, the rules are called when certain events occur, for example, when a message is put, a message expires, or a duplicate message arrives. The rules then determine how the queue handles these events.

The base set of queue rules are defined in this class, which should be extended if a solution wishes to alter queue behaviour.

Queues hold messages in a queue store, typically this would be a persistent type of storage, such as a disk drive, but it does not need to necessarily be so. The queue maintains an index entry for each message held in its queue store. The index entry consists of state information for the message, such as whether it is locked or unlocked. Also, certain message fields are stored in the index entry, these are known as index fields. The default index fields are, message unique ID, MQSeries message ID, MQSeries correlation ID, and message priority. These fields are stored because they are present in most messages, and storing the fields in memory yields faster message searching.

The `indexEntry()` rule is called whenever an index entry is created. This occurs whenever a new message is put onto the queue or at queue activation time when the queue reads any messages left in its queue store from a previous session. The rule allows a solution to alter the index entry when it is created, one use for this would be to add an extra field or fields into the index, thus improving message search times.

Queues maintain a use count, this is incremented when the queue is activated, and likewise decremented when the queue is closed. Also, the use count is incremented when a remote queue manager connects to a queue. The use count is decremented when the channel and transporter used to create this connection are destroyed. The `useCountChanged()` rule is called every time the use count is changed.

The messages held on the queue can be protected by an authenticator and cryptor. The messages can also be compressed by using a compressor. Together, the authenticator, cryptor, and compressor are known as the attributes of the queue, and they are defined by specifying an appropriate **MQeAttribute** object to be associated with the queue. The `attributeChange()` rule is called whenever an attempt is made to replace the queue's attribute.

Note: Changing a queue's attribute when it already holds messages stored using another attribute, may cause message loss, since the message may not be recoverable using the new attribute.

If a message has remained on the queue for a length of time greater than the queue's expiry interval, or if a message exceeds its own expiry interval then the `messageExpired()` rule is called. This rule then determines what happens to the message, but typically the message would be either deleted, or placed on a 'dead letter queue'.

Package
com.ibm.mqe

Method summary

Method	Purpose
addListener	This rule is called when a message listener is attached to the queue.
attributeChange	This rule is called when an attempt is made to change the queue's attribute. The attribute defines the authenticator, cryptor and compressor used on the queue. Messages are stored using the attributes defined.
browseMessage	The rule decides whether to allow the message to be included in the set of messages returns to the application that issued the browse request.
deleteMessage	This rule is called when a delete message operation takes place.
duplicateMessage	This rule is called if a duplicate message is put onto the queue.
filterMessage	This rule is called when a message filter is being applied during a GetMessage, BrowseMessage or DeleteMessage method call.
getMessage	This rule is called when a message is found that satisfies a get message request. The rule can decide whether the message should be allowed to satisfy the get message request.
getPendingMessage	This rule is called when a get pending message request is received.
indexEntry	This rule is called whenever an index entry is created
messageExpired	This rule is called when a message has exceeded either the queue's expiry interval, or it's own expiry interval. The rule then decides whether to expire the message, and what to do with the message if it has expired.
putMessage	This rule is called when a put message request is made.
queueActivate	This rule is called when the queue is activated.
queueClosed	This rule is called when the queue is closed.
removeListener	This rule is called when a message listener is removed from a queue.
resetMessageLock	This rule is called when a request is made to reset the lock state on a message. The message is reset to an unlocked state. This function can only be performed by MQSeries Administration. By throwing an exception the rule can prevent the reset from occurring.
undo	This rule is called when an undo operation occurs and it determines whether the message is to be included in the set of messages processed by the undo operation.
useCountChanged	This rule is called every time the use count is changed

MQeQueueRule addListener

Syntax

```
public void addListener( MQeMessageListenerInterface listener,
                       MQeFields filter) throws Exception
```

MQeQueueRule

Description

This rule is called when a message listener is attached to a queue. By throwing an exception the rule is able to reject the add listener request.

Parameters

- listener** A reference to the object that is subscribing to MQSeries Everyplace message events. The object must implement **MQeMessageListenerInterface**.
- filter** *null* or an **MQeFields** object containing message fields. A value of *null* means that the listener wishes to receive events for all messages on the queue.
- Specifying an **MQeFields** object containing message fields means that the listener is only interested in events concerning messages whose fields match those contained in the filter.

Return values

none

Exceptions

none

Examples

```
class exampleQueueRules extends MQeQueueRule
{
    ...
    /* This rule logs the addition of a message listener */
    public void addListener( MQeMessageListenerInterface listener,
                           MQeFields filter ) throws Exception
    {
        log( MQe_Log_Information, Event_Queue_AddMsgListener,
            "Added listener on queue " +
            ((MQeQueue)owner).getQueueManagerName() + "+" +
            ((MQeQueue)owner).getQueueName() );
    }

    public void queueActivate()
    { /* create a new log file */
        try
        {
            logFile = new LogToDiskFile( "\\log.txt" );
        }
        catch( Exception e )
        {
            e.printStackTrace( System.err );
        }
    }

    public void queueClose()
    { /* close log file */
        logFile.close();
    }
    ...
}
```

Related functions

- removeListener

MQeQueueRule attributeChange

Syntax

```
public void attributeChange( MQeAttribute attribute ) throws Exception
```

Description

This method is called when an attempt is made to change the queue's attribute. The attribute defines the authenticator, cryptor and compressor used on the queue. All messages are stored using the queue's attribute.

By throwing an exception the rule is able to reject the attribute change request.

Parameters

attribute *null*, or an **MQeAttribute** object that defines the authenticator, cryptor and compressor that will be used on the queue, if the change is allowed. *null* means that no attribute will be used on the queue.

Return values

none

Exceptions

Example

```
class exampleQueueRules extends MQeQueueRule
{
    ...
    /* This rule only allows the queue's attribute to be changed if it was */
    /* not previously set */
    /* The queue object is the owner of the rule */
    public void attributeChange( MQeAttribute attribute ) throws Exception
    {
        if ( ((MQeQueue) owner).getQueueAttribute() != null )
            throw new MQeException( Except_Rule, "Attribute already set" );
    }
    ...
}
```

MQeQueueRule browseMessage

Syntax

```
public boolean browseMessage( MQeMsgObject msg,
                             long confirmId ) throws Exception
```

Description

A Browse Messages operation matches zero or more messages held on a queue. This method is called for every message matched. The rule decides whether to allow the message to be included in the set of messages returned to the application that issued the browse request.

If this rule throws an exception then the browse operation will be terminated.

Parameters

msg An **MQeMsgObject** containing the message being browsed.

confirmId A long value that is the confirm Id used on this browse operation. The confirm Id is used to restore messages in the event of a failure

MQeQueueRule

Return values

A boolean value that denotes whether to allow this message to be included in the set of messages returned to the application that issued the browse request. .

True Allow the message to be included

False Do not allow the message to be included

Exceptions

Examples

```
class exampleQueueRules extends MQeQueueRule
{
    ...
    /* This rule only allows messages of type 'OrderResponse' to be browsed */
    public boolean browseMessage( MQeMsgObject msg,
                                long confirmID ) throws Exception
    {
        /* get message type field */
        String msgType = msg.getAscii( "MsgType" );
        /* what message type is it? */
        if ( msgType.equals( "OrderResponse" ) )
            return (true); /* allow browse */
        else
            return (false); /* don't allow browse */
    }
    ...
}
```

MQeQueueRule deleteMessage

Syntax

```
public void deleteMessage( MQeFields filter ) throws Exception
```

Description

This rule is called when a delete message operation takes place. By throwing an exception the rule can reject the delete message request.

Parameters

filter An **MQeFields** object containing a message filter. The filter must contain the message's UID for the delete operation to be successful.

Return values

none

Exceptions

Example

```
class exampleQueueRules extends MQeQueueRule
{
    ...
    /* This rule requires that the filter contain a password */
    public void deleteMessage( MQeFields filter ) throws Exception
    {
        if ( filter != null && filter.contains( "Password" ) )
        {
            String pswd = filter.getAscii( "Password" );
            if ( pswd.equals( "12345678" ) )
            { /* remove password from filter */
                filter.delete( "Password" );
                return;
            }
        }
        else
    }
```

```

        throw new MQeException( Except_Rule, "Incorrect password" );
    }
    throw new MQeException( Except_Rule, "No Password" );
}
...
}

```

MQeQueueRule duplicateMessage

Syntax

```

public void duplicateMessage( MQeMsgObject msg,
                             long confirmId ) throws Exception

```

Description

This rule is called if a duplicate message is put onto the queue.

Parameters

msg	An MQeMsgObject containing the duplicate message.
confirmId	A long value that is the confirm Id used on this put operation. The confirm Id is used to restore messages in the event of a failure

Return values

none

Exceptions

Examples

```

class exampleQueueRules extends MQeQueueRule
{
    ...
    /* This rule logs the duplicate message exception */
    public void duplicateMessage( MQeMsgObject msg,
                                  long confirmID ) throws Exception
    {
        /* get message UID */
        MQeFields msgUID = msg.getMsgUIDFields();
        /* log the duplicate message exception */
        log( MQe_Log_Warning, Event_Queue_Duplicate,
            msgUID.getAscii( MQe.Msg_OriginQMgr ) + " + " +
            msgUID.getLong( MQe.Msg_Time ) );
    }

    public void queueActivate()
    { /* create a new log file */
        try
        {
            logFile = new LogToDiskFile( "\\log.txt" );
        }
        catch( Exception e )
        {
            e.printStackTrace( System.err );
        }
    }

    public void queueClose()
    { /* close log file */
        logFile.close();
    }

    ...
}

```

MQeQueueRule

MQeQueueRule filterMessage

Syntax

```
public boolean filterMessage( int what, MQeFields filter )
```

Description

This rule is called when a message filter is being applied during a GetMessage, BrowseMessage or DeleteMessage call. The queue rule checks to make sure certain fields exist in the filter, and that they contain certain values. The rule then uses this information to allow or reject the operation.

Parameters

what An integer, indicating which operation is being performed:

- 0 = browse
- 1 = get
- 2 = delete

filter An MQeFields object. *null* or the search filter used to restrict the operation to a subset messages on the queue.

Return values

true Allow the requested operation to go ahead
false Reject the requested operation

Exceptions

None

Examples

```
public class ExampleQueueRule extends MQeQueueRule {  
    ...  
    public boolean filterMessage( int what , MQeFields filter ) {  
        boolean result = true ; // Allow all operations by default, regardless of filter.  
        return result ;  
    }  
    ...  
}
```

Related Functions

- getMessage
- browseMessage
- deleteMessage

MQeQueueRule getMessage

Syntax

```
public boolean getMessage( MQeMsgObject msg,  
                           long confirmId ) throws Exception
```

Description

This rule is called when a message is found which satisfies a get message request. The rule can decide whether that message should be allowed to satisfy the get message request.

If the rule does not allow the message to satisfy the get message request then the queue will search for another message that satisfies the request.

By throwing an exception the rule can terminate the get message request.

Parameters

msg	An MQeMsgObject containing a message that satisfies the get message request..
confirmId	A long value that is the confirm Id used on this get request. The confirm Id is used to restore messages in the event of a failure

Return values

none

Exceptions

none

Examples

```

class exampleQueueRules extends MQeQueueRule
{
    ...
    /* This rule disallows a get request on a message if the message states */
    /* that it can only be browsed */
    public boolean getMessage( MQeMsgObject msg,
                               long confirmId ) throws Exception
    { /* does the message contain a field which states what operations are */
      /* allowed upon it */
      if ( msg.contains( "AllowableOperations" ) )
      {
          String allowedOperations = msg.getAscii( "AllowableOperations" );
          /* if the message states that it can only be browsed disallow the get */
          if ( allowedOperations.equals( "BrowseOnly" ) )
              return (false);
      }
      return (true);
    }
    ...
}

```

Related functions

- putMessage

MQeQueueRule getPendingMessage**Syntax**

```

public void getPendingMessage( String queueManagerName,
                               MQeFields filter,
                               long confirmId ) throws Exception

```

Description

An MQSeries Everyplace queue manager is able to collect messages destined for itself from its home-server through a home-server queue. The home-server stores messages destined for its clients in one or more store-and-forward queues. The home-server queue contacts its home-server's store-and-forward queues using the MQeStoreAndForwardQueue.getPendingMessage() method. Any pending messages for that MQSeries Everyplace queue manager are then returned.

This rule is called when a get pending message request is received.

Parameters**queueManagerName**

A String containing the name of the MQSeries Everyplace queue manager that initiated the get pending message request.

filter

null or an **MQeFields** object containing a message filter,

MQeQueueRule

used to match any pending messages. *null* means return the first available message addressed to the queue manager specified in `queueManagerName`.

confirmId A long value that is the confirm ID for this operation. The confirmID is used to restore messages in the event of a failure.

Return values
none

Exceptions

Examples

```
class exampleQueueRules extends MQeQueueRule
{
    ...
    /* This rule requires that the filter contain a password */
    /* (For this rule to work correctly in would be necessary to override */
    /* MQeHomeServerQueue so that the message filter sent to the Store & */
    /* Forward Queue was non-null) */
    public void getPendingMessage( String queueManagerName, MQeFields filter,
                                  long confirmId ) throws Exception
    {
        if ( filter != null && filter.contains( "Password" ) )
        {
            String pswd = filter.getAscii( "Password" );
            if ( pswd.equals( "123456878" ) )
            { /* remove password from filter */
                filter.delete( "Password" );
                return;
            }
        }
        throw new MQeException( Except_Rule, "No Password" );
    }
    ...
}
```

MQeQueueRule indexEntry

Syntax

```
public void indexEntry( MQeFields entry,
                       MQeMsgObject msg ) throws Exception
```

Description

This rule is called when the queue creates an index entry. This occurs when a new message is put onto the queue, and when the queue is activated, if it still holds messages from a previous session.

The index entry contains state information about the message, along with certain index fields that are held to enable faster message searching. These fields are:

- MQe Unique ID (MQe.Msg_OriginQMgr + MQe.Msg_Time)
- MQ Series Message ID (MQe.Msg_ID)
- MQ Series Correlation ID (MQe.Msg_CorrelID)
- Message Priority(MQe.Msg_Priority)

Parameters

entry A String containing the name of the MQSeries Everyplace queue manager that initiated the get pending message request.

filter	An MQeFields object containing a blank index entry. The rule can add fields to this object, if it wishes.
msg	An MQeMsgObject containing the message for which an index entry is being created.

Return values
none

Exceptions

Examples

```
class exampleQueueRules extends MQQueueRule
{
    ...
    /* if the message contains a customer number field - then add this field */
    /* to the message's index entry. */
    /* This will enable faster message searching */
    public void indexEntry( MQeFields entry,
                           MQeMsgObject msg ) throws Exception
    {
        if ( msg.contains( "Cust_No" ) )
            entry.copy( msg, true, "Cust_No" );
    }
    ...
}
```

MQQueueRule messageExpired

Syntax

```
public boolean messageExpired( MQeFields entry,
                              MQeMsgObject msg ) throws Exception
```

Description

This rule is called when a message has exceeded either the queue's expiry interval, or it's own expiry interval. The check to see whether the message exceeded the expiry intervals is made every time the message is accessed.

The rule then decides whether to expire the message, and what subsequently happens to the message, if it has expired.

Parameters

entry	An MQeFields object containing the index entry for the message that has expired.
msg	An MQeMsgObject containing the message that has expired.

Return values

A boolean value which denotes whether to expire the message.

true	The message has expired and can be deleted
false	The message has not expired

Exceptions

Examples

```
class exampleQueueRules extends MQQueueRule
{
    ...
    /* This rule puts a copy of any expired messages to a Dead Letter Queue */
    public boolean messageExpired( MQeFields entry,
                                   MQeMsgObject msg ) throws Exception
    {
```

MQeQueueRule

```
/* Get the reference to the Queue Manager */
MQeQueueManager qmgr = MQeQueueManager.getReference(
    ((MQeQueue)owner).getQueueManagerName() );
/* need to set re-send flag so that put of message to new queue isn't */
/* rejected /
msg.putBoolean( MQe.Msg_Resend, true );
/* if the message contains an expiry interval field - remove it */
if ( msg.contains( MQe.Msg_ExpireTime )
    msg.delete( MQe.Msg_ExpireTime );
/* put message onto dead letter queue */
qmgr.putMessage( null, "DEAD.LETTER.QUEUE", msg, null, 0 );
/* return true & the message will be deleted from the queue */
return (true);
}
...
}
```

MQeQueueRule putMessage

Syntax

```
public void putMessage( MQeMsgObject msg,
    long confirmID ) throws Exception
```

Description

This rule is called when a put message request is made. By throwing an exception the rule can prevent the message being put onto the queue.

Parameters

msg	An MQeMsgObject containing the msg to be put onto the queue.
confirmID	A long value containing the confirm ID for this operation. The confirm ID is used to restore locked messages in the event of a failure.

Return values

none

Exceptions

Examples

```
class exampleQueueRules extends MQeQueueRule
{
    ...
    /* This rule blocks a message Put if the message priority is less than 5 */
    public void putMessage( MQeMsgObject msg, long confirmID ) throws Exception
    {
        if ( (msg.contains( MQe.Msg_Priority )) &&
            (msg.getByte( MQe.Msg_Priority ) < 5) )
            throw new MQeException( Except_Rule, "Priority too low" );
    }
    ...
}
```

Related functions

- getMessage

MQeQueueRule queueActivate

Syntax

```
public void queueActivate()
```

Description

This rule is called when the queue is activated.

Parameters

none

Return values

none

Exceptions

none

Examples

```

class exampleQueueRules extends MQeQueueRule
{
    ...
    /* This rule logs the activation of the queue */
    public void queueActivate()
    {
        try
        {
            logFile = new LogToDiskFile( "\\log.txt );
            log( MQe_Log_Information, Event_Queue_Activate, "Queue " +
                ((MQeQueue)owner).getQueueManagerName() + " + " +
                ((MQeQueue)owner).getQueueName() + " active" );
        }
        catch( Exception e )
        {
            e.printStackTrace( System.err );
        }
    }

    public void queueClose()
    { /* close log file */
        logFile.close();
    }
    ...
}

```

Related functions

- queueClose

MQeQueueRule queueClose**Syntax**

```
public void queueClose()
```

Description

This rule is called when the queue is closed.

Parameters

none

Return values

none

Exceptions

none

Examples

```

class exampleQueueRules extends MQeQueueRule
{
    ...
    /* This rule logs the closure of the queue */
    public void queueClose()
    {
        try
        {

```

MQeQueueRule

```
        log( MQe_Log_Information, Event_Queue_Closed, "Queue " +
            ((MQeQueue)owner).getQueueManagerName() + " + " +
            ((MQeQueue)owner).getQueueName() + " closed" );
        /* close log file */
        logFile.close();
    }
    catch ( Exception e )
    {
        e.printStackTrace( System.err );
    }
}

public void queueActivate()
{
    try
    {
        logFile = new LogToDiskFile( "\\log.txt );
        log( MQe_Log_Information, Event_Queue_Activate, "Queue " +
            ((MQeQueue)owner).getQueueManagerName() + " + " +
            ((MQeQueue)owner).getQueueName() + " active" );
    }
    catch( Exception e )
    {
        e.printStackTrace( System.err );
    }
} ...
}
```

Related functions

- queueActivate

MQeQueueRule removeListener

Syntax

```
public void removeListener( MQeMessageListenerInterface listener,
                           MQeFields filter ) throws Exception
```

Description

This rule is called when a message listener is removed from a queue. By throwing an exception the rule can prevent the listener from being removed.

Parameters

- | | |
|-----------------|--|
| listener | The object that is subscribing to MQSeries Everyplace message events. The object must implement MQeMessageListenerInterface . |
| filter | <i>null</i> or an MQeFields object containing a message filter. The filter must match the filter used in the add listener command that created this listener. |

Return values

none

Exceptions

Examples

```
class exampleQueueRules extends MQeQueueRule
{
    ...
    /* This rule logs the removal of the message listener */
    public void removeListener( MQeMessageListenerInterface listener,
                               MQeFields filter ) throws Exception
    {
```

```

log( MQe_Log_Information, Event_Queue_RemoveMsgListener,
    "Removed listener on queue " +
    ((MQeQueue)owner).getQueueManagerName() + " " +
    ((MQeQueue)owner).getQueueName() );
}

public void queueActivate()
{ /* create a new log file */
  try
  {
    logFile = new LogToDiskFile( "\\log.txt );
  }
  catch( Exception e )
  {
    e.printStackTrace( System.err );
  }
}

public void queueClose()
{ /* close log file */
  logFile.close();
}

...
}

```

Related functions

- addListener

MQeQueueRule resetMessageLock**Syntax**

```
public void resetMessageLock( MQeFields filter ) throws Exception
```

Description

This rule is called when a request is made to reset the lock state on a message. The message is reset to an unlocked state. This function can only be performed by MQe Administration. By throwing an exception the rule can prevent the reset from occurring.

Parameters

filter An **MQeFields** object containing a message filter. This filter is used to match the message having its lock state reset

Return values

none

Exceptions**Examples**

```

class exampleQueueRules extends MQeQueueRule
{
  ...
  /* This rule logs the message lock reset */
  public void resetMessageLock( MQeFields filter ) throws Exception
  { /* get message UID */
    if ( filter.contains( MQe.Msg_Time ) &&
        filter.contains( MQe.Msg_OriginQMgr ) )
    {
      String originQMgr = filter.getAscii( MQe.Msg_OriginQMgr );
      long timeStamp    = filter.getLong( MQe.Msg_Time );

      log( MQe_Log_Information, Event_Queue_ResetMessageLock,
          "Message " + originQMgr + ":" + timeStamp + " on queue " +
          ((MQeQueue)owner).getQueueManagerName() + " " +

```

MQeQueueRule

```
        ((MQeQueue)owner).getQueueName() + " has been reset" );
    }
}

public void queueActivate()
{ /* create a new log file */
    try
    {
        logFile = new LogToDiskFile( "\\log.txt );
    }
    catch( Exception e )
    {
        e.printStackTrace( System.err );
    }
}

public void queueClose()
{ /* close log file */
    logFile.close();
}

...
}
```

MQeQueueRule undo

Syntax

```
public boolean undo( MQeFields filter ) throws Exception
```

Description

An undo message operation matches zero or more messages held on a queue. This rule is called for each message matched, and the rule determines whether the message is to be included in the set of messages processed by the undo operation.

By throwing an exception the rule can terminate the undo operation.

Parameters

- filter** An **MQeFields** object containing a message filter. Only messages matching the filter are included in the undo operation.
- confirmId** A long value that was the confirm Id used by the operation which is being undone. All messages matching this confirm Id are restored to their previous state

Return values

A boolean value denoting whether this message will be included in the set of messages processed by the undo operation.

- true** Include this message in those processed by the undo operation
- false** Do not include this message in those processed by the undo operation

Exceptions

Examples

```
class exampleQueueRules extends MQeQueueRule
{
    ...
    /* This rule logs the message reset */
    public boolean undo( MQeFields filter ) throws Exception
    { /* get message UID */
        if ( filter.contains( MQe.Msg_Time ) &&
```

```

        filter.contains( MQe.Msg_OriginQMgr ) )
    {
        String originQMgr = filter.getAscii( MQe.Msg_OriginQMgr );
        long timeStamp    = filter.getLong( MQe.Msg_Time );

        log( MQe_Log_Information, Event_Queue_ResetMessageLock,
            "Message " + originQMgr + ":" + timeStamp + " on queue " +
            ((MQeQueue)owner).getQueueManagerName() + " + " +
            ((MQeQueue)owner).getQueueName() + " has been reset" );
    }
    return (true);
}

public void queueActivate()
{ /* create a new log file */
    try
    {
        logFile = new LogToDiskFile( "\\log.txt );
    }
    catch( Exception e )
    {
        e.printStackTrace( System.err );
    }
}

public void queueClose()
{ /* close log file */
    logFile.close();
}

...
}

```

MQeQueueRule useCountChanged

Syntax

```
public void useCountChanged( int useCount ) throws Exception
```

Description

This rule is called when a queue's use count changes. The use count is a measure of the number of users attached to the queue. The use count changes when the queue is activated or closed, and when remote queue managers connect to, or disconnect from, the queue using an **MQeTransporter**.

Parameters

useCount the queue's current use count.

Return values

none

Exceptions

Examples

```

class exampleQueueRules extends MQeQueueRule
{
    ...
    /* do not allow the use count to exceed ten */
    public void useCountChanged( int useCount ) throws Exception
    {
        if ( useCount == 10 )
            throw new MQeException( Except_Rule, "Too many users" );
    }
    ...
}

```

MQEventLogInterface

All MQSeries Everyplace log handlers must implement this interface.

Package **com.ibm.mqe**

Method summary

Method	Purpose
activate	Activates the event log handler
close	Terminates the event log function and performs any cleanup as appropriate
logOutput	Outputs data to the event log

MQEventLogInterface activate

Syntax

```
public void activate( String logName ) throws Exception
```

Description

Is called to activate the event log handler

Parameters

logName A String used to identify this event log

Return values

none

MQEventLogInterface close

Syntax

```
public void close( )
```

Description

Called to close the event log handler and to perform any cleanup as necessary

Parameters

none

Return values

none

MQEventLogInterface logOutput

Syntax

```
public void logOutput( String data )
```

Description

Called by MQSeries Everyplace to output a message to the event log handler

Parameters

data The data to be logged

Return values

none

MQeMessageListenerInterface

This interface must be implemented by all objects that wish to receive MQeMessage events.

Package **com.ibm.mqe**

Method summary

Method	Purpose
messageArrived	Event handler for MQeMessageEvent.MessageArrived events. This event is generated when a message arrives on a queue.

MQeMessageListener messageArrived

Syntax

```
public void messageArrived( MQeMessageEvent msgEvent )
```

Description

This method is called on all listening objects when an MQeMessageEvent.MessageArrived event is generated.

Parameters

msgEvent An MQeMessageEvent object containing details of the newly arrived message

Return values

none

Exceptions

none.

Example

```
class MyMQeApplication extends MQSeries Everyplace
                        implements MQeMessageListenerInterface
{
    ...
    public void messageArrived( MQeMessageEvent e )
    {
        ...
        if ( e.getQueueName().equals("MY.QUEUE") )
            MQeFields msgFields = e.getMsgFields(); /* get msg info */
        ...
    }
    ...
}
```

MQeRunListInterface

Two lists of MQSeries Everyplace applications can be passed to an MQSeries Everyplace queue manager as part of the parameter set passed to it when it is activated. The applications contained in the first list are invoked once the queue manager is active. The applications contained in the second list are invoked when the queue manager receives a close request.

All applications should implement MQeRunListInterface, but it is not mandatory.

Package **com.ibm.mqe**

Method summary

Method	Purpose
activate	called by the queue manager to activate the application.

MQeRunListInterface activate

Syntax

```
public Object activate( Object owner,
                      Hashtable loadTable,
                      MQeFields setupData ) throws Exception;
```

Description

Two lists of MQSeries Everyplace applications can be passed to a MQSeries Everyplace queue manager as part of the parameters passed when the queue manager is activated. The first list contains applications that are invoked once the queue manager is active. The second list contains applications that are invoked when the queue manager receives a close request (when the MQeQueueManager.close() method is called).

If the applications contained in the queue manager parameters implement **MQeRunListInterface**, then the queue manager calls this method to activate the application, and pass any set-up information for the application that is contained in the queue manager parameters.

Applications are not forced to implement MQeRunListInterface, but if they do not, the application is just invoked and no set-up information is passed to it

An application that is invoked when a queue manager is activated should return from this method as quickly as possible to allow the queue manager to continue. The application should initialize itself on a different thread from the one on which it is called. The application is responsible for the management of this thread.

An application that is invoked on queue manager close can block the queue manager from shutting down by not returning from this method.

Parameters

owner This is the object that owns the application. Usually this is the MQe queue manager that invoked the application.

loadTable A java.util.Hashtable object that can be used to share data between the applications invoked by the queue manager. All of the applications invoked by the queue manager have a reference to this table.

setupData An **MQeFields** object containing application setup data. This data must have been included in the parameters passed to the queue manager when the queue manager was activated. See the sample INI file below for an example of this.

Return values

An object reference - this is not currently used.

Exceptions

none

Example

Example of an application being launched when the queue manager is activated

```
public class ExampleApp extends MQe implements MQeRunListInterface,
                                               Runnable,
                                               MQeMessageListenerInterface
{
    Thread th = null;
    MQeQueueManager qmgr = null;
    ...
    /* Called by the Queue Manager to activate the application */
    public Object activate( Object owner, Hashtable loadTable,
                          MQeFields setupData )
    {
        qmgr = (MQeQueueManager)owner; /* QMgr is owner of the application */
        processSetupData( setupData ); /* Process the setup information */
        th = new Thread( this ); /* Create a new thread to listen */
        th.start(); /* for incoming messages */
        return (null); /* return control to the QMgr */
    }

    public void run()
    {
        try
        { /* Create a message listener for incoming messages */
            qmgr.addMessageListener( this, "MyQueue", null );
            /* Loop indefinitely keeping application alive */
            while( true );
        }
        catch ( Exception e )
        {
            e.printStackTrace( System.err );
        }
    }
    ...
}
```

Example of an application being launched when the queue manager receives a close request

```
public class ExampleCloseApp extends MQe implements MQeRunListInterface
{
    MQeQueueManager qmgr = null;
    ...
    /* Called by the Queue Manager to activate the application */
    public Object activate( Object owner, Hashtable loadTable,
                          MQeFields setupData )
    {
        qmgr = (MQeQueueManager)owner; /* QMgr is owner of the application */
        performAction(); /* Perform some action */
        /* don't return control to the QMgr until application has finished */
        return (null);
    }
}
```

MQeRunListInterface

Sample Queue Manager INI file

```
* Sample Queue Manager INI file

* Queue Manager setup info
[QueueManager]
* Name for this Queue Manager
(ascii)Name=ServerQMGr8082

* Registry setup info
[Registry]
* QueueManager Registry type
(ascii)LocalRegType=com.ibm.mqe.registry.MQePrivateSession
* Location of the registry
(ascii)DirName=d:\development\Rename\Classes\ServerQMGr8082\Registry
* Registry access PIN
(ascii)PIN=12345678

* List of applications to launched at Queue Manager activation-time
[ActivateAppList]
(ascii)App1=examples.queuemanager.TestMQeApp
(ascii)App2=examples.administration.AdminApp

* Setup info for App1 - the data in this section is passed to the application
[App1]
(ascii)DefaultMsgPriority = 7
(long)Timeout = 30000

* Setup info for App2 - the data in this section is passed to the application
[App1]
(ascii)DefaultQueueName=AdminReplyQueue
```

MQeSecurityInterface

This is an optional interface that may be implemented by a Java Security manager . The interface methods allow the security manager to authorize or reject the call. MQSeries Everyplace trace handlers must implement this interface.

Package **com.ibm.MQe**

Method summary

Method	Support
alias	Called whenever a class alias is added or removed
channelCommand	Called whenever a channel command is about to be processed by a channel
newAdapter	Called whenever an adapter definition is about to be defined
mapFileDescriptor	Called whenever a mapping of a file descriptor is about to be set

MQeSecurityInterface alias

Syntax

```
public void alias( String from, String to ) throws Exception
```

Description

Called whenever an alias is about to be set or removed

Parameters

from A String defining the class alias

to A String defining the class name for the alias, or *null* if the alias is to be removed

Return values

none

Exceptions

SecurityException Request was rejected

Example

MQeSecurityInterface channelCommand

Syntax

```
public void channelCommand( String command ) throws Exception
```

Description

Called whenever a channel command is about to be processed

Parameters

command A string containing the channel command

Return values

none

Exceptions

SecurityException request was rejected

Example

MQeSecurityInterface

MQeSecurityInterface newAdapter

Syntax

```
public void newAdapter( String destination ) throws Exception
```

Description

Called whenever a new adapter definition is about to be set

Parameters

destination A string containing the destination for this adapter. A typical Destination would be of the form:
Tcpip:127.0.0.1:8080

Return values

none

Exceptions

SecurityException request was rejected

Example

MQeSecurityInterface mapFileDescriptor

Syntax

```
public void mMapFileDescriptor( String fileDesc, Object newDesc )  
                               throws Exception
```

Description

Called whenever a file descriptor map is to be set

Parameters

fileDesc A string containing the file descriptor that is to be mapped
newDesc A string containing the file descriptor that is the mapped descriptor

Return values

none

Exceptions

SecurityException Request was rejected

Example

MQeTraceInterface

All MQSeries Everyplace trace handlers must implement this interface.

Package **com.ibm.mqe**

Method summary

Method	Purpose
activate	Called to activate the trace handler
addMessage	Adds a new trace message template
addMessageBundle	Adds a bundle of trace message templates
close	Called to close the trace handler and perform any cleanup as appropriate
getMessage	Return the message template for a given message number
traceMessage	Called to write a trace message to the output stream

MQeTraceInterface activate

Syntax

```
public void activate ( String title, String resource )
```

Description

Called to activate the trace handler

Parameters

title A String to be used as a title for this trace handler or *null*

resource A String used to identify the resource bundle to use for this trace handler

Return values

none

MQeTraceInterface traceMessage

Syntax

```
public String traceMessage( String prefix,
                           int msgNumber,
                           Object insert )
```

Description

Called by MQSeries Everyplace to output a trace message via the trace handler

Parameters

prefix The calling object name and instance number

msgNumber An integer containing the trace message number to be used to find the message template

insert Any inserts to be applied to the message template

Return values

A String containing the expanded trace message text

MQeTraceInterface

MQeTraceInterface addMessage

Syntax

1. `public void addMessage (int msgNumber, String msgText) throws Exception`
2. `public void addMessage (String msgText) throws Exception`

Description

Called to add a new trace message template to the trace handler. The template should be of the form:

```
static final Object[][] contents = {
/*-----*/
/* System messages */
/* ' ' message */
/* 'i' Information */
/* 'w' Warning */
/* 'e' Error */
/* 'd' Debug */
/*
/* Application messages */
/* ' ' message */
/* 'I' Information */
/* 'W' Warning */
/* 'E' Error */
/* 'D' Debug */
/*
/* Modifier */
/* ':' no modification applied */
/* ';' RESERVED for create/destroy object */
/* '+' Log this message via the Log interface */
/* '' ignore - Do not display this message */
/*
/* Mwsage number */
/* "[nnnn]:" syntax for message number of this message */
/*
/* Example: */
/* "e+[01000]:Error #0 occurred" */
/* "I:[01010]:All is wonderful" */
/*-----*/
}
```

Parameters

- msgNumber** An integer containing the trace message number to be used to identify the message template
- msgText** A String containing the trace message template

Return values

none

MQeTraceInterface addMessageBundle

Syntax

```
public void addMessageBundle( String msgBundle ) throws Exception
```

Description

Is called to add a bundle of templates to the trace handler. The templates in the bundle should be of the form:

```
static final Object[][] contents = {
/*-----*/
/* System messages */
/* ' ' message */
/* 'i' Information */
/* 'w' Warning */
/*

```


MQeTracelInterface

```
/* 'e'      Error      */
/* 'd'      Debug     */
/*
/* Application messages
/* ' '      message    */
/* 'I'      Information */
/* 'W'      Warning    */
/* 'E'      Error      */
/* 'D'      Debug     */
/*
/* Modifier
/* ':'      no modification applied
/* ';'      RESERVED for create/destroy object
/* '+'      Log this message via the Log interface
/* ''      ignore - Do not display this message
/*
/* Mwsage number
/* "[nnnn]:" syntax for message number of this message
/*
/* Example:
/* "e+[01000]:Error #0 occurred"
/* "I:[01010]:All is wonderful"
/*-----*/
```

Parameters

msgBundle

A String identifying the bundle of trace messages templates to be added

Return values

none

Standard trace messages

```
static final Object[] [] contents = {
/*-----*/
/* System messages
/* ' '      message    */
/* 'i'      Information */
/* 'w'      Warning    */
/* 'e'      Error      */
/* 's'      Security    */
/* 'd'      Debug     */
/*
/* Application messages
/* ' '      message    */
/* 'I'      Information */
/* 'W'      Warning    */
/* 'E'      Error      */
/* 'S'      Security    */
/* 'D'      Debug     */
/*
/* Modifier
/* ':'      no modification applied
/* ';'      RESERVED for create/destroy object
/* '+'      Log this message via the Log interface
/* ''      ignore - Do not display this message
/*
/* Mwsage number
/* "[nnnn]:" syntax for message number of this message
/*
/* Example:
/* "e+[01000]:Error #0 occurred"
/* "I:[01010]:All is wonderful"
/*-----*/
```

MQeTraceInterface

```

/* common messages */
{
  "1",    "d:[00001]:Created"    },
{
  "2",    "d:[00002]:Destroyed" },
{
  "3",    "d:[00003]:Close"    },
{
  "4",    "w:[00004]:Warning:#" },
{
  "5",    "e:[00005]:Error:#"  },
{
  "6",    "i:[00006]:Command:#" },
{
  "7",    "i:[00007]:Waiting"  },
{
  "8",    "i:[00008]:# input byte count=#" },
{
  "9",    "i:[00009]:# output byte count=#" },
/* com.ibm.MQe.MQeLoader */
{
  "10",   "d:[00010]:loadClass #"    },
{
  "11",   "d:[00011]:Loaded (bytes) #" },
{
  "12",   "d:[00012]:Resolved Class #" },
{
  "13",   "d:[00013]:DropClass #0"   },
/* com.ibm.MQe.MQeChannel & ChannelManager */
{
  "20",   "d:[00020]:ActivateMaster"  },
{
  "21",   "d:[00021]:ActivateSlave"   },
{
  "22",   "d:[00022]:ActivateSlave Channel ID=#0" },
{
  "23",   "d:[00023]:Close Channel ID=#0" },
{
  "24",   "d:[00024]:SlaveResponse"   },
{
  "25",   "d:[00025]:SlaveResponse Channel ID=#0" },
{
  "26",   "i:[00026]:Timeout channel ID=#0" },
{
  "27",   "i:[00027]:Forwarding to #0" },
{
  "28",   "i:[00028]:ID=#0, Command=#1" },
/* com.ibm.MQe.MQeChannelListener */
{
  "30",   "i:[00030]:Starting Listener #0" },
{
  "31",   "i:[00031]:Stopping Listener"   },
{
  "32",   "d:[00032]:Starting Slave"     },
{
  "33",   "d:[00033]:Stopping Slave"     },
{
  "34",   "d:[00034]:Timer interval"     },
/* com.ibm.MQe.MQeAttribute */
{
  "40",   "d:[00040]:Authenticator #0"   },
{
  "41",   "d:[00041]:Compressor #0"     },
{
  "42",   "d:[00042]:Cryptor #0"        },
{
  "43",   "d:[00043]:Attribute(Rule).equals=#0" },
{
  "44",   "d:[00044]:Attribute Change #0" },
{
  "45",   "d:[00045]:TargetRegistry=#0" },
{
  "46",   "s:[00046]:Secure Chnl State=pending, KeyObject=#0" },
/* com.ibm.MQe.MQeTransporter */
{
  "50",   "d:[00050]:#0 PID=#1"         },
{
  "51",   "d:[00051]:#0 made persistent PID=#1" },
{
  "52",   "d:[00052]:#0 Message Request for Queue '#1'" },
{
  "53",   "d:[00053]:GetPendingMessage for Queue Manager '#0'" },
/* ***** Adapters ***** */
/* com.ibm.MQe.Adapters.MQeIniFileAdapter & MQeDisk...Adapter */
{
  "110",  "d:[00110]:Object #0 - saved"   },
{
  "111",  "d:[00111]:Object #0 - loaded"  },
{
  "112",  "d:[00112]:Object #0 - Selected" },
{
  "113",  "d:[00113]:Object #0 - matched" },
{
  "114",  "d:[00114]:Object #0 - deleted" },
/* com.ibm.MQe.Adapters.MQeTcpipAdapter */
{
  "200",  "d:[00200]:File descriptor '#0'" },
{
  "201",  "d:[00201]:Socket pending"      },
{
  "202",  "d:[00202]:Control '#0'"      },
/* com.ibm.MQe.Adapters.MQeTcpipHttpAdapter */
{
  "203",  "d:[00203]:Read Header"        },
{
  "204",  "d:[00204]:Header: #0"        },
{
  "205",  "d:[00205]:Header length=#0"   },
{
  "206",  "d:[00206]:Write Header"       },
{
  "207",  "d:[00207]:Read Content-length=#0" },
{
  "208",  "d:[00208]:Readln #0"        },
...

```

```
...
```

```
};
```

The full list of messages can be found in the `examples.trace.MQeTraceResource.java` source file in the `trace` subdirectory of the `examples` directory.

MQeTraceInterface close

Syntax

Description

Parameters

Return values

MQeTraceInterface getMessage

Syntax

```
public String getMessage( int msgNumber )
```

Description

Called to get the String that corresponds to the trace message number supplied in the **msgNumber** parameter

Parameters

msgNumber The number of the trace message String to be returned

Return values

A String containing the trace message template

MQeTraceInterface

Chapter 3. Classes in com.ibm.mqe.administration

This section contains detailed information about the following MQSeries Everyplace classes and interfaces:

Table 13. Classes in package com.ibm.mqe.Administration

Class or interface name	Purpose
MQeAdminQueueAdminMsg	Used to manage queues of type MQeAdminQueue
MQeConnectionAdminMsg	Class used to manage connections of type MQeConnectionDefinition
MQeHomeServerQueueAdminMsg	Used to manage queues of type MQeHomeServerQueue
MQeQueueAdminMsg	Used to manage MQSeries Everyplace local queues of type MQeQueue
MQeQueueManagerAdminMsg	Used to manage queue managers of type MQeQueueManager
MQeRemoteQueueAdminMsg	Used to manage remote queues of type MQeRemoteQueue
MQeStoreAndForwardQueueAdminMsg	Used to manage queues of type MQeStoreAndForwardQueue

MQeAdminQueueAdminMsg

This class is used to manage queues of type MQeAdminQueue. The class extends **MQeQueueAdminMsg** and provides an implementation for managing administration queues.

This queue is used to provide transparent local/remote administration of MQSeries Everyplace managed resources.

Package **com.ibm.mqe.administration**

This class is a descendant of **MQeQueueAdminMsg**

Constants and variables

MQeHomeServerQueueAdminMsg provides the following constants and variables in addition to those provided and inherited by **MQeQueueAdminMsg**:

Queue characteristics

QtimerInterval;

Process outstanding administration messages after interval (milliseconds)
(long)

```
public final static String Queue_QtimerInterval;
```

MQeConnectionAdminMsg

This class is used to manage connections of type MQeConnectionDefinition.

The class extends **MQeAdminMsg** and provides the implementation for managing connections. The following actions are applicable on connections:

- **Action_Create**
- **Action_Delete**
- **Action_Inquire**
- **Action_InquireAll**
- **Action_Update**

Connections define how one queue manager establishes a connection to another queue manager. The main characteristics associated with a connection are:

The type of channel to use

The following types of channel are provided:

MQeChannel

Channel for Client to Server or Server to Server

MQPeerChannel

Channel for Peer to Peer

A file descriptor that contains the communications adapter and its parameters

The following adapters are provided:

MQeTcpipLengthAdapter

Simple TCPIP adapter

MQeTcpipHistoryAdapter

TCPIP adapter that provides persistent connections and data compression

MQeTcpipHttpAdapter

HTTP adapter

A file descriptor for an http connection to server "192.168.0.1" on port 8081 would be specified as:

```
com.ibm.mqe.Adpaters.MQeTcpipHttpAdapter:192.168.0.1:8081
```

or if an alias of "Network" had been set up for the HTTP adapter then the following file descriptor could be used:

```
Network:192.168.0.1:8081
```

Package **com.ibm.mqe.administration**

This class is a descendant of **MQeAdminMsg**

Constants and variables

MQeQueueAdminMsg provides the following constants and variables in addition to those provided by **MQeAdminMsg**:

Additional actions

AddAlias

Add connection aliases

MQeConnectionAdminMsg

```
public final static int Action_AddAlias
```

RemoveAlias

Remove connection aliases

```
public final static int Action_RemoveAlias
```

Connection characteristics

Adapters (fields array)

Set of adapters used to connect to a target queue managers.

Note: In release one only one adapter is allowed.

```
public final static String Con_Adapters
```

The following fields define those allowed in each element of the **adapters** array:

AdapterFileDesc (ascii)

The adapter file descriptor

AdapterAsciiParm (ascii)

Parameters for the adapter

AdapterEncodedParm (byte array)

Encoded parameters

AdapterOptions (ascii)

Options for the adapter

```
public final static String Con_Adapter
public final static String Con_AdapterAsciiParm
public final static String Con_AdapterEncodedParm
```

Aliases

Set of aliases for this connection (ascii array)

```
public final static String Con_Aliases
```

Channel

Channel class (ascii) - the type of channel that this connection should use. For example:

```
com.ibm.mqe.MQeChannel
com.ibm.mqe.MQePeerChannel
```

the value can also be null meaning that this is a local connection.

```
public final static String Con_Channel
```

Description

Description of the connection (unicode)

```
public final static String Con_Description
```

Method summary

Method	Purpose
create	Sets up the administration message to perform the Action_Create action
update	Sets up the administration message to perform the Action_Update action.

MQeConnectionAdminMsg create

Syntax

```
public void create( String adapter, String parameters,
                  String options, String channel
                  String description ) throws Exception
```

Description

Sets up the administration message to perform the Action_Create action. This version of create adds a simple connection definition for a connection that has one adapter.

Parameters

adapter	File descriptor for the adapter
parameters	Adapter parameters
options	Adapter options
channel	Type of channel to use
description	Description of connection

Return values

none

Exceptions

java.lang.Exception	Various
----------------------------	---------

Example

```
class MyApplication
{
    ...
    MQeConnectionAdminMsg con = new MQeConnectionAdminMsg()
    con.setName("ServerQM123");
    con.create("Network:127.0.0.1:8081",
              null,
              null,
              "DefaultChannel",
              "Con to MQeServer" );
    MQeConnectionAdminMsg con2 = new MQeConnectionAdminMsg()
    con2.setName("ServletQM123");
    con2.create("Network:127.0.0.1:8081",
               "/servlet/MQe",
               null,
               "DefaultChannel",
               "Con to MQeServlet" );
    ...
}
```

MQeConnectionAdminMsg update

Syntax

```
public void update( String adapter, String parameters,
                  String options, String channel
                  String description ) throws Exception
```

Description

Sets up the administration message to perform the Action_Update action. This version of update replaces an existing connection definition with a simple connection definition for a connection that has one adapter.

Parameters

adapter	File descriptor for the adapter
----------------	---------------------------------

MQeConnectionAdminMsg

parameters	Adapter parameters
options	Adapter options
channel	Type of channel to use
description	Description of the connection

Return values

none

Exceptions

java.lang.Exception	Various
----------------------------	---------

Example

```
class MyApplication
{
    ...
    MQeConnectionAdminMsg con = new MQeConnectionAdminMsg()
    con.setName("ServerQM123");
    con.update("Network:127.0.0.1:8082",
              null,
              null,
              "DefaultChannel",
              "Con to MQeServer" );
    ...
}
```

MQeHomeServerQueueAdminMsg

This class is used to manage queues of type MQeHomeServerQueue. The class extends **MQeRemoteQueueAdminMsg** and provides an implementation for managing home server queues

This queue is used in a client to retrieve (using a background thread) pending messages from its *home* queue on its *HomeServer*.

Package **com.ibm.mqe.administration**

This class is a descendant of **MQeRemoteQueueAdminMsg**

Constants and variables

MQeHomeServerQueueAdminMsg provides the following constants and variables in addition to those provided and inherited by **MQeRemoteQueueAdminMsg**:

Queue characteristics

QtimerInterval;

Process pending messages after interval (milliseconds) (long)

```
public final static String Queue_QtimerInterval;
```

MQeQueueAdminMsg

This class is used to manage MQSeries Everyplace local queues of type MQeQueue. The class extends **MQeAdminMsg** and provides an implementation for managing local queues. The following actions are applicable on local queues:

- Action_Create
- Action_Delete
- Action_Inquire
- Action_InquireAll
- Action_Update
- Action_AddAlias
- Action_RemoteAlias

Local queues, as the name suggest are local to the owning queue manager. A file descriptor must be set that details where and how the queue is stored. It is formed from two parts, an adapter and parameters to the adapter. The following adapters are provided:

MQeDiskFieldsAdapter

File based adapter for **MQeFields** objects

MQeMemoryFieldsAdapter

Memory based adapter for **MQeFields** objects

For example, if alias MsgLog is set to MQeDiskFieldsAdapter then to store messages at d:\ServerQM123\Queues, the file descriptor would be:

MsgLog:d:\ServerQM123\Queues

Queues allow several characteristics to be set that are not used by the base local queue. These characteristics are made available to a user replaceable queue rules class that can make use of them. For instance Queue_MaxQSize can be set but is not checked by MQeQueue. It is the responsibility of the queues rules class to perform maximum queue size validation.

This class acts as the base class for managing other types of queues. For instance MQeRemoteQueueAdminMsg derives from this class and handles management of remote queues.

Package **com.ibm.mqe.Administration**

This class is a descendant of **MQeAdminMsg**

Constants and variables

MQeQueueAdminMsg provides the following constants and variables in addition to those provided by **MQeAdminMsg**:

Additional actions

AddAlias

Add queue aliases

```
public final static int    Action_AddAlias;
```

RemoveAlias

Remove queue aliases

```
public final static int    Action_RemoveAlias;
```

Queue characteristics

Active Indicates that the queue is active (boolean, Read Only)

```
public final static String Queue_Active
```

CreationDate

Date queue was created (long Read Only). Time in milliseconds since midnight Jan 1, 1970 GMT

```
public final static String Queue_CreationDate
```

CurrentSize

Current queue depth (int, Read Only)

```
public final static String Queue_CurrentSize
```

Description

Description of the queue (unicode)

```
public final static String Queue_Description
```

Expiry Messages on queue expire *n* milliseconds after being stored on the queue (long)

```
public final static String Queue_Expiry
```

FileDesc

File descriptor - the location where queue is stored (ascii)

Once set the file descriptor cannot be changed.

The file descriptor is formed from two parts

- An adapter
- The adapter's parameters

For example, if alias MsgLog is set to MQeDiskFieldsAdapter then to store messages at d:\ServerQM123\Queues, the filedescrptor would be:

```
MsgLog:d:\ServerQM123\Queues
```

```
public final static String Queue_FileDesc
```

MaxMsgSize

Maximum length of messages allowed on the queue (int)

```
public final static String Queue_MaxMsgSize
```

MaxQSize

Maximum number of messages allowed on the queue (int)

NoLimit

```
public final static String Queue_MaxQSize
```

```
public final static int Queue_NoLimit
```

Mode Type of queue - Is the queue synchronous or asynchronous.

```
public final static String Queue_Mode
```

Asynchronous

Queue is asynchronous

```
public final static byte Queue_Asynchronous
```

Synchronous

Queue is synchronous

```
public final static byte Queue_Synchronous
```

Priority

Default priority for messages if not already specified in the message (byte) (min = 0, max = 9)

MQeQueueAdminMsg

public final static String Queue_Priority

QAliasNameList

Set of alias names for this queue (ascii array)

public final static String Queue_QAliasNameList

QMgrName

Name of the queue manager that owns the queue (ascii). Once set the queue manager name cannot be changed.

public final static String Queue_QMgrName

Queue security Characteristics

These fields can only be changed when the queue has 0 messages and is not active.

AttrRule

Name of queue attribute rules class (ascii)

public final static String Queue_AttrRule

Authenticator

Name of authenticator class (ascii)

public final static String Queue_Authenticator

Compressor

Name of compressor class (ascii)

public final static String Queue_Compressor

Cryptor

Name of cryptor class (ascii)

public final static String Queue_Cryptor

TargetRegistry

Target registry type (byte)

public final static String Queue_TargetRegistry

One of the following:

RegistryNone

public final static byte Queue_RegistryNone

RegistryQMgr

public final static byte Queue_RegistryQMgr

RegistryQueue

public final static byte Queue_RegistryQueue

Rule Name of queue rules class (ascii)

public final static String Queue_Rule

Constructor summary

Constructor	Purpose
MQeQueueAdminMsg	There are two constructors. One creates and initializes a default MQeQueueAdminMsg. The other takes the name of the queue that is to be managed.

Method summary

Constructor	Purpose
addAlias	Sets up an administration message to perform the Admin_AddAlias action.
removeAlias	Sets up an administration message to perform the Admin_RemoveAlias action.
setName	Sets the name of the queue that the action is to be performed against.

MQeQueueAdminMsg

Syntax

1. `public MQeQueueAdminMsg() throws Exception`
2. `public MQeQueueAdminMsg(String qMgrName, String queueName)`

Description

There are 2 constructors.

1. This version creates and initializes a default MQeAdminMsg
2. This version takes the name of the queue that is to be managed

Parameters

- qMgrName** Queue manager name
queueName Queue name

Return Values

A new MQeQueueAdminMsg

Exceptions

- java.lang.Exception** Various

Example

```
class MyApplication
{
    MQeQueueAdminMsg aMsg = new MQeQueueAdminMsg( "ExampleQM", "ExampleQ" );
}
```

MQeQueueAdminMsg addAlias

Syntax

`public void addAlias(String aliasName) throws Exception`

Description

Sets up an administration message to perform the Admin_AddAlias action. A queue can have no aliases, one or several aliases. This method can be called more than once to allow multiple aliases to be added in one administration message.

Parameters

- aliasName** Alias name of queue

Return values

none

Exceptions

- java.lang.Exception** Various

MQQueueAdminMsg

Example

```
class MyApplication
{
    ...
    // Add aliases to a queue
    MQQueueAdminMsg msg = new MQQueueAdminMsg();
    msg.setName( "ExampleQM", "ExampleQ" );

    // Set the action required and its parameters
    // into the message
    msg.addAlias( "PayrollQ" );
    msg.addAlias( "Branch1PayrollQ" );
    ...
}
```

MQQueueAdminMsg removeAlias

Syntax

```
public void removeAlias( String aliasName ) throws Exception
```

Description

Sets up an administration message to perform the Admin_RemoveAlias action. This action removes the named alias from the queue. This method may be called more than once to allow multiple aliases to be removed using one administration message.

Parameters

aliasName Alias name of queue

Return values

none

Exceptions

java.lang.Exception Various

Example

```
class MyApplication
{
    ...
    // Add aliases to a queue
    MQQueueAdminMsg msg = new MQQueueAdminMsg();
    msg.setName( "ExampleQM", "ExampleQ" );

    // Set the action required and its parameters
    // into the message
    msg.removeAlias( "PayrollQ" );
    msg.removeAlias( "Branch1PayrollQ" );
    ...
}
```

MQQueueAdminMsg setName

Syntax

```
public void setName( String qMgrName, String queueName ) throws Exception
```

Description

Sets the name of the queue that the action is to be performed on.

Parameters

qMgrName Name of the queue manager that owns the queue
queueName Name of the queue

Return values

none

Exceptions**java.lang.Exception** Various**Example**

```
class MyApplication
{
    ...
    // Delete a queue
    MqeFields parms = new MqeFields();

    // Set the action required and its parameters
    // into the message
    msg.delete( parms );
    msg.setName( "ExampleQM", "ExampleQ" );
    ....
}
```

MQeQueueManagerAdminMsg

This section describes the Java class used to create a basic MQeQueueManagerAdminMsg.

The class extends **MQeAdminMsg** and provides the implementation for managing MQSeries Everyplace queue managers. The following actions are applicable on queue managers:

- **Action_Inquire**
- **Action_InquireAll**
- **Action_Update**

Note: **Create** and **Delete** actions are not supported on queue managers as a queue manager has to be in place and have an administration queue initialized on it before administration can take place, and only one queue manager per Java virtual machine is supported.

Package **com.ibm.mqe.administration**

This class is a descendant of **MQeAdminMsg**

Constants and variables

MQeQueueManagerAdminMsg provides the following constants and variables in addition to those provided and inherited by **MQeAdminMsg**

Queue manager characteristics

ChnlAttrRules

Channel attribute rules.

```
public final static String QMgr_ChnlAttrRules
```

ChnlTimeout

Maximum amount of time in milliseconds that a channel remains open (long).

```
public final static String QMgr_ChnlTimeout
```

Connections

Connections known by queue manager (ascii array - Read only)

```
public final static String QMgr_Connections
```

Description

Description of queue manager (unicode)

```
public final static String QMgr_Description
```

Queues

Queues known by queue manager (fields array - Read Only)

```
public final static String QMgr_Queues
```

QueueName

Queue name (ascii)

```
public final static String QMgr_QueueName
```

QueueQMgrName

Queue manager name (ascii)

```
public final static String QMgr_QueueQMgrName
```

QueueType

Queue type (ascii)

MQeQueueManagerAdminMsg

```
public final static String QMgr_QueueType
```

Rules User replaceable rules which control the capability of the queue manager (ascii).

```
public final static String QMgr_Rules
```

MQeRemoteQueueAdminMsg

This class is used to manage remote queues of type MQeRemoteQueue. The class extends **MQeQueueAdminMsg** and provides an implementation for managing remote queues.

There are two types of remote queue:

Synchronous

When a request is made to a remote queue that is set for synchronous access, a channel is opened to the node where the queue is local. Hence all actions on a synchronous queue are shipped to the location where the queue is local at the time the request is made. Hence networking capability must be available between the remote queue and local queue at the time the request is made.

Asynchronous

When a request is made to a remote queue that is set for asynchronous access, the message is temporarily stored in the remote queue. Based on user replaceable rules, the message is moved from the remote queue to the location where the queue is local at some point in the future. Hence remote asynchronous queues require a file descriptor that describes where the messages are stored before they are moved.

Package **com.ibm.mqe.administration**

This class is a descendant of **MQeQueueAdminMsg**

Constants and variables

MQeRemoteQueueAdminMsg provides the following constants and variables in addition to those provided and inherited by **MQeQueueAdminMsg**:

Queue characteristics

Transporter

Name of the transporter class to use (ascii)

DefaultTransporter

```
public final static String Queue_Transporter
public final static String Queue_DefaultTransporter
```

TransporterXOR

Allow transporter to use xor compression (boolean).

This provides intelligent compression of messages when they are moved across a network by XORing each field in a message with a field of the same name (if it exists) in the previous message that the transporter moved.

```
public final static String Queue_TransporterXOR
```

MQeStoreAndForwardQueueAdminMsg

This class is used to manage queues of type MQeStoreAndForwardQueue. The class extends MQeRemoteQueueAdminMsg and provides an implementation for managing store and forward queues.

This type of queue is used in intermediate nodes to hold messages that are just passing through i.e. that are not destined for any queues on this system.

Package **com.ibm.mqe.administration**

This class is a descendant of **MQeRemoteQueueAdminMsg**

Constants and variables

MQeStoreAndForwardQueueAdminMsg provides the following constants and variables in addition to those provided and inherited by **MQeRemoteQueueAdminMsg**:

Additional actions

AddQueueManager

Add queue manager

```
public final static int    Action_AddQueueManager;
```

RemoveQueueManager

Remove queue manager

```
public final static int    Action_RemoveQueueManager;
```

Remote queue characteristics

QMgrNameList

List of queue manager targets handled by this store and forward queue messages for the target queue managers are temporarily stored in this queue until they are collected by the target queue manager or until communication can be established.

```
public final static String    Queue_QMgrNameList
```

Method summary

Method	Purpose
addQueueManager	Sets up an administration message to perform the Admin_AddQueueManager action
removeQueueManager	Sets up an administration message to perform the Admin_RemoveQueueManager action

MQeStoreAndForwardQueueAdminMsg addQueueManager

Syntax

```
public void addQueueManager( String targetQMgrName ) throws Exception
```

Description

Sets up an administration message to perform the Admin_AddQueueManager action. A queue may have one or more target queue managers. This method may be called more than once to allow multiple targets be added in one administration message.

Parameters

MQeStoreAndForwardQueueAdminMsg

targetQMgrName
Target queue manager name

Return values
none

Exceptions

java.lang.Exception Various

Example

```
class MyApplication
{
    ...
    // Add target queue managers to a S&F queue
    MQeStoreAndForwardQueueAdminMsg msg =
        new MQeStoreAndForwardQueueAdminMsg();
    msg.setName( "ExampleQM", "ExampleQ" );

    // Set the action required and its parameters
    // into the message
    msg.addQueueManager( "Client129345" );
    msg.addQueueManager( "Client129387" );
    ...
}
```

MQeStoreAndForwardQueueAdminMsg removeQueueManager

Syntax

public void removeQueueManager(String targetQMgrName) throws Exception

Description

Sets up an administration message to perform the Admin_RemoveQueueManager action. This method may be called more than once to allow multiple targets be removed in one administration message.

Parameters

targetQMgrName target queue manager name

Return values
none

Exceptions

java.lang.Exception Various

Example

```
class MyApplication
{
    ...
    // Remove target queue managers from S&F queue
    MQeStoreAndForwardQueueAdminMsg msg =
        new MQeStoreAndForwardQueueAdminMsg();
    msg.setName( "ExampleQM", "ExampleQ" );

    // Set the action required and its parameters
    // into the message
    msg.removeQueueManager( "Client129345" );
    msg.removeQueueManager( "Client129387" );
    ...
}
```

Chapter 4. Classes in com.ibm.mqe.attributes

This section contains detailed information about the following MQSeries Everyplace classes:

Table 14. Classes in package com.ibm.mqe.attributes

Class name	Purpose
**MQe3DESCryptor	Provides mechanisms for 3DES encryption
MQeDESCryptor	Provides mechanisms for DES encryption
MQeGenDH	Creates an MQeDHk.java file from which solution unique MQeDHk class objects can be created
MQeLocalSecure	Provides a simple local security service
MQeLZWCompressor	Provides mechanisms for LZW compression
**MQeMARSCryptor	Provides mechanisms for MARS encryption
MQeMAttribute	Provide simple message-level protection
**MQeMTrustAttribute	Provides more advanced message-level protection
**MQeRC4Cryptor	Provides mechanisms for RC4 encryption
**MQeRC6Cryptor	Provides mechanisms for RC6 encryption
MQeRleCompressor	Provides mechanisms for Run Length encoded compression
**MQeWTLSertAuthenticator	Provides mechanisms for mini-certificate authentication
MQeXORCryptor	Provides mechanisms for XOR encryption

Note: Classes marked ** are available only in the high security version of MQSeries Everyplace Version 1.1.

MQe3DESCryptor

Note: This class is only available in the high security version of MQSeries Everyplace Version 1.1.

This class is used to create a triple DES cryptor object that, when used by an attribute object, provides the attribute object with the mechanisms to perform triple DES encryption. Attribute objects are associated with channel and **MQeFields** objects.

Package **com.ibm.mqe.attributes**

This class is a descendant of **MQeCryptor**

MQe3DESCryptor

Syntax

```
public MQe3DESCryptor( )
```

Description

Constructs an MQe3DESCryptor object

Parameters

none

Return values

none

Exceptions

MQeException	Except_S_Cipher, "cip3DES, wrong cipher or key"
---------------------	---

Example

```
try
{
    MQe3DESCryptor tripledес = new MQe3DESCryptor();
    MQeAttribute tripledесA = new MQeAttribute(null, tripledес, null);
    ...
}
catch ( Exception e )
{
    ...
}
```

Related functions

- **MQeCryptor**
- **MQeAttribute**
- **MQeLocalSecure**
- **MQeMAttribute**
- **MQeMTrustAttribute**

MQeDESCryptor

This class is used to create a DES Cryptor object that, when used by an attribute object, provides the attribute object with the mechanisms to perform DES encryption. Attribute objects are associated with channel and **MQeFields** objects.

Package **com.ibm.mqe.attributes**

This class is a descendant of **MQeCryptor**

MQeDESCryptor

Syntax

```
public MQeDESCryptor( )
```

Description

Constructs an MQeDESCryptor object

Parameters

none

Return values

none

Exceptions

MQeException

Except_S_Cipher, "cipDES, wrong cipher or key"

Example

```
try
{
    MQeDESCryptor des = new MQeDESCryptor();
    MQeAttribute desA = new MQeAttribute(null, des, null);
    ...
}
catch ( Exception e )
{
    ...
}
```

Related functions

- **MQeCryptor**
- **MQeAttribute**
- **MQeLocalSecure**
- **MQeMAttribute**
- **MQeMTrustAttribute**

MQeGenDH

This class is used to create an MQeDhK.java file from which solution unique MQeDhK class objects can be created.

Package **com.ibm.mqe.attributes**

This class is a descendant of MQe

Method summary

Method	Purpose
main	Invokes a utility to generate a new MQDhK.java file
genParams	Generates a new DH pair and uses the new pair to create a new MQeDhK.java file

Part of the data flowed in *secure channel* establishment is Diffie Hellman partial key data. This data is used to generate a shared secret key, derivatives of which are subsequently used, to protect the confidentiality of the channel data, by the channel cryptor's encrypt and decrypt methods . The example shows how to use the utility to create a 512 bit Diffie Hellman key pair. To make this available to MQe, the resulting MQeDhK.java file must be compiled and installed as part of the com.ibm.mqe.attributes package.

MQeGenDH main

Syntax

```
java com.ibm.mqe.attributes.MQeGenDH <parameter1><parameter2>
```

Description

Invokes a utility to generate a new DH key pair and create a new MQeDhK.java file.

Parameters

<parameter 1>	Optional, DH parameter length, defaults to 512
<parameter 2>	Optional, Trace class name, e.g. examples.awt.AwtMQeTrace, defaults to use System Console

Return values

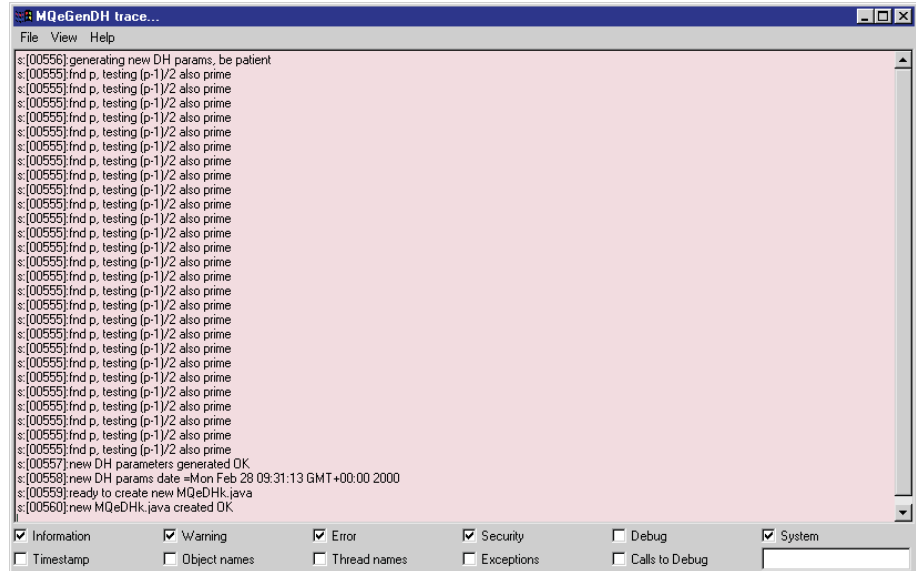
none - new MQeDhK file created in the current directory

Exceptions

none

Example

```
java        com.ibm.mqe.attributes.MQeGenDH        512        examples.awt.AwtMQeTrace
```



MQeGenDH genParams

Syntax

```
public void genParams(int length)
```

Description

Invokes the utility to generate a new DH key pair and create a new MQeDhK.java file.

Parameters

length Parameter bit length

Return Values

none - new MQeDhK file created in the current directory

Exceptions

none

MQeLocalSecure

This class is used to create a LocalSecure object that provides a simple local security service enabling a using application to apply a given attribute's (cryptor and compressor) components to protect local data

Package **com.ibm.mqe.attributes**

This class is a descendant of **MQe**

Constructor summary

Constructor	Purpose
MQeLocalSecure	Constructs an MQeLocalSecure object

Method summary

Method	Purpose
open	Enables the using application to identify the target File
read	Reads, unprotects and returns data from the target file
write	Protects and writes the given data to the target file

MQeLocalSecure

Syntax

```
public MQeLocalSecure( )
```

Description

Constructs an MQeLocalSecure object

Parameters

none

Return values

none

Exceptions

none

Example

```
MQeLocalSecure ls = new MQeLocalSecure( );
```

Related functions

- **MQeAttribute**

MQeLocalSecure open

Syntax

```
public void open( String directory,  
                 Object fileName )
```

Description

Sets the target filename.

Parameters

directory A string identifying the target File Directory

fileName A string identifying the target file name

Return Values

none

Exceptions

none

Related functions

- write
- read

MQeLocalSecure read

Syntax

```
Public byte[] read( MQeAttribute attr,
                  String localCipherKey ) throws Exception
```

Description

Reads and unprotects data from the given target filename

Parameters

attr **MQeAttribute** to be applied to unprotect data

localCipherKey

Password or passphrase String to be used to unprotect data

Return Values

none

Exceptions

MQeException

Except_S_InvalidAttribute, "no cryptor"

Except_S_InvalidAttribute, "illegal cryptor"

Except_S_InvalidAttribute, "illegal authenticator or compressor"

MQe.Except_Data, "wrong cipher"

Exception

java.io

Example

```
try
{
    MQeDESCryptor des = new MQeDESCryptor( );
    MQeAttribute desA = new MQeAttribute( null, des, null);
    MQeLocalSecure ls = new MQeLocalSecure( );
    ls.open( ".\\", "TestSecureData.txt" );
    String outData = byteToAscii( ls.read( A,
                                         "It_is_a_secret" ) );
    Trace ( "i: unprotected data = " + outData);
    ...
}
catch ( Exception e )
{
    ...
}
```

MQeLocalSecure write

Syntax

MQeLocalSecure

```
Public void write ( byte[] data,  
                  MQeAttribute attr,  
                  String localCipherKey) throws Exception
```

Description

Protects data and writes it to the given target filename.

Parameters

data	Data to protect
attr	MQeAttribute to be applied to protect data
localCipherKey	Password or passphrase String to be used to protect data

Return Values

none

Exceptions

MQeException	Except_S_InvalidAttribute, "no cryptor" Except_S_InvalidAttribute, "illegal cryptor" Except_S_InvalidAttribute, "illegal authenticator or compressor"
Exception	java.io

Example

```
try  
{  
    MQeDESCryptor des = new MQeDESCryptor( );  
    MQeAttribute desA = new MQeAttribute( null, des, null);  
    MQeLocalSecure ls = new MQeLocalSecure( );  
    ls.open( ".\\", "TestSecureData.txt" );  
    ls.write( asciiToByte( "0123456789abcdef..." ),  
            desA, "It_is_a_secret" );  
    ...  
}  
catch ( Exception e )  
{  
    ...  
}
```

MQeLZWCompressor

This class is used to create an LZW Compressor object that, when used by an attribute object, provides the attribute object with the mechanisms to perform LZW compression. Attribute objects are associated with channel and **MQeFields** objects.

Package **com.ibm.mqe.attributes**

This class is a descendant of **MQeCompressor**

MQeLZWCompressor

Syntax

```
public MQeLZWCompressor( )
```

Description

Constructs an MQeLZWCompressor object

Parameters

none

Return values

none

Exceptions

none

Example

```
try
{
    MQeLZWCompressor lzw = new MQeLZWCompressor();
    MQeAttribute lzwA    = new MQeAttribute(null, null, lzw);
    ...
}
catch ( Exception e )
{
    ...
}
```

Related functions

- **MQeCompressor**
- **MQeAttribute**
- **MQeLocalSecure**
- **MQeMAttribute**
- **MQeMTrustAttribute**

MQeMARSCryptor

Note: This class is only available in the high security version of MQSeries Everyplace Version 1.1.

This class is used to create a MARS cryptor object that, when used by an attribute object, provides the attribute object with the mechanisms to perform MARS encryption. Attribute objects are associated with channel and **MQeFields** objects.

Package **com.ibm.mqe.attributes**

This class is a descendant of **MQeCryptor**

MQeMARSCryptor

Syntax

```
public MQeMARSCryptor( )
```

Description

Constructs an MQeMARS cryptor object

Parameters

none

Return values

none

Exceptions

none

Example

```
try
{
    MQeMARSCryptor mars = new MQeMARSCryptor();
    MQeAttribute marsA = new MQeAttribute(null, mars, null);
    ...
}
catch ( Exception e )
{
    ...
}
```

Related functions

- **MQeCryptor**
- **MQeAttribute**
- **MQeLocalSecure**
- **MQeMAttribute**
- **MQeMTrustAttribute**

MQeMAttribute

This class is used to create an attribute object enabling simple message-level protection when attached to a message.

Package **com.ibm.mqe.attributes**

This class is a descendant of **MQeAttribute**

Constructor summary

Constructor	Purpose
MQeMAttribute	Constructs an MQeMAttribute object

Method summary

Method	Purpose
decodeData	Decrypt and/or decompress the supplied data
encodeData	Compress and/or encrypt the supplied data

MQeMAttribute

Syntax

1. public MQeMAttribute()
2. public MQeMAttribute(MQeAuthenticator authenticator, MQeCryptor cryptor, MQeCompressor compressor) throws Exception

Description

Constructs an **MQeMAttribute** object

Parameters

- authenticator** *null* or an object reference to an **MQeAuthenticator** object
- cryptor** An object reference to a symmetric **MQeCryptor** object (**MQeDESCryptor**, **MQe3DESCryptor**, **MQeRC4Cryptor**, **MQeRC6Cryptor** or **MQeMARSCryptor**)
- compressor** *null* or an object reference to a **MQeCompressor** object (**MQeRleCompressor** or **MQeLZWCompressor**)

Return values

none

Exceptions

- MQeException** Except_ Not Supported, "invalid authenticator"
- Except_ Not Supported, "invalid cryptor"
- Except_ Not Supported, "invalid compressor"

Example

```
class MySampleClass extends MQe
{
  /* application on initiating QueueManager: */
  /* -prepare to use MQeMAttribute with Rle Compressor */
}
```

MQeMAttribute

```
/* and DES Cryptor with key = It_is_a_secret */

MQeKey localkey      = new MQeKey();
localkey.setLocalKey( "It_is_a_secret");
MQeDESCryptor des    = new MQeDESCryptor();
MQeRleCompressor rle = new MQeRleCompressor();
MQeMAttribute protMAttr = new MQeMAttribute( null, des, rle );
protMAttr.setKey( localkey );
/* construct Message and protect with the MQeMAttribute */
MQeMessageObj MsgObj = new MQeMessageObject();
MsgObj.setAttribute( protMAttr ); /* add test message data */
MsgObj.putAscii("MsgData", "0123456789abcdef...");
trace ("i: input message data = " + MsgObj.getAscii("MsgData" ) );
/* assume MQeQueueManager instance initQM started, PutMessage */
initQM.putMessage( targetQMGrName, targetQName, MsgObj ,null, 0);

...
...//
...//.
...//.
/* application on recipient QueueManager: */
/* -prepare to use MQeMAttribute with key = It_is_a_secret */
MQeKey localkey      = new MQeKey();
localkey.setLocalKey( "It_is_a_secret");
MQeDESCryptor des    = new MQeDESCryptor();
MQeRleCompressor rle = new MQeRleCompressor();
MQeMAttribute protMAttr = new MQeMAttribute( null, des, rle );
protMAttr.setKey( localkey );
/* assume MQeQueueManager instance recipQM started, GetMessage */
MQeMsgObject MsgObj = recipQM.getMessage(thisQMGrName,
                                          thisQName, null, protMAttr, 0);
trace ("i: output message data = " + MsgObj.getAscii( "MsgData" ) );

}
```

Related functions

MQeMAttribute

MQeMAttribute decodeData

Syntax

```
public byte[] decodeData( MQeChannel channel,
                        byte data[],
                        int offset,
                        int count ) throws Exception
```

Description

Is called to decode (decrypt and/or decompress) the bytes referenced by **data**, **offset** and for length **count**.

Note: This method is intended for internal use and is not normally called by applications.

Parameters

channel	<i>null</i> , not used
data	An object reference to a byte array containing the data to be decoded
offset	An integer index specifying the start byte in the data array
count	An integer count of the number of bytes to decode

Return Values

Decoded data

Exceptions

MQeException	Except_Data, "data tampering detected"
	Except_S_NoPresetKeyAvailable

MQeMAttribute encodeData

Syntax

```
public byte[] encodeData( MQeChannel channel,
                        byte data[],
                        int offset,
                        int count ) throws Exception
```

Description

Encodes (encrypts and/or compresses) the bytes referenced by *data*, **offset** and for length **count**.

Note: This method is intended for internal use and is not normally called by applications.

Parameters

channel	<i>null</i> , not used
data	An object reference to a byte array containing the data to be encoded
offset	An integer index specifying the start byte in the data array
count	An integer count of the number of bytes to encode

Return Values

none

Exceptions

MQeException	Except_ Not Supported, "invalid cryptor"
	Except_ Not Supported, "invalid compressor"
	Except_S_NoPresetKeyAvailable

MQeMTrustAttribute

Note: This class is only available in the high security version of MQSeries Everyplace Version 1.1.

This class is used to create an attribute object enabling message-level protection of message objects in such a way that :

- Validation of the originator's (ISO9796) digital signature enables the recipient to establish the message's origin (nonrepudation)
- Message confidentiality is protected using the attribute's cryptor
- Message integrity is validated
- Message data can only be restored by the intended recipient

Package **com.ibm.mqe.attributes**

This class is a descendant of **MQeAttribute**

Constructor summary

Constructor	Purpose
MQeMTrustAttribute	Constructs an MQeMTrustAttribute object

Method summary

Method	Purpose
decodeData	Decodes the supplied data
encodeData	Encodes the supplied data.
setHomeServer	Sets address of home server/alternative MQeNode
setPrivateRegistry	Sets active private registry
setPublicRegistry	Sets active public registry

MQeMTrustAttribute

Syntax

1. public MQeMTrustAttribute()
2. public MQeMTrustAttribute(MQeAuthenticator authenticator, MQeCryptor cryptor, MQeCompressor compressor) throws Exception

Description

Constructs an MQeMTrustAttribute object

Parameters

- authenticator** *null*, not used
- cryptor** An object reference to a symmetric **MQeCryptor** object (**MQeDESCryptor**, **MQe3DESCryptor**, **MQeRC4Cryptor**, **MQeRC6Cryptor** or **MQeMARSCryptor**)
- compressor** *null*, not used

Return values

none

Exceptions

MQeException

Except_ Not Supported, "invalid cryptor"

Example

```

class MySampleClass extends MQe
{
  /* application on initiating QueueManager: */
  /* use MQeMTrustAttribute to protect a message between */
  /* pre-reg'd initiator 'Bruce1' and recipient 'Bruce8' */
  /* assume initiator's QueueManager initQM started */
  /* setup MTrustAttribute */
  MQeMARSCryptor mars = new MQeMARSCryptor( );
  MQeMTrustAttribute msgA = new MQeMTrustAttribute( null, mars, null );

  /* setup instantiate & activate sender (Bruce1) PrivReg */
  String EntityName = "Bruce1";
  String EntityPIN = "12345678";
  Object KeyRingPassword = "It_is_a_secret";
  MQePrivateRegistry sendreg = new MQePrivateRegistry( );
  sendreg.activate( EntityName, "./MQeNode_PrivateRegistry",
                  EntityPIN,KeyRingPassword, null, null );

  /* set target entity's registry name into sender's reg */
  sendreg.setTargetRegistryName("Bruce8");
  /* set MTrustAttribute s PrivateRegistry = sendreg */
  msgA.setPrivateRegistry( sendreg );
  /* instantiate and activate Public Registry which has */
  /* (or gets) MiniCert of intended recipient (Bruce8) */
  MQePublicRegistry pr = new MQePublicRegistry( );
  pr.activate( "MQeNode_PublicRegistry", "/" );
  /* set MTrustAttribute's PublicRegistry & HomeServer */
  msgA.setPublicRegistry(pr);
  msgA.setHomeServer( MyHomeServer + ":8081" );

  /* create message object and add some test data */
  MQeMsgObject msgObj = new MQeMsgObject( );
  msgObj.putArrayOfByte( "TestData",
                        asciiToByte("0123456789abcdef....") );
  /* protect with MQeMTrustAttribute and PutMessage */
  msgObj.setAttribute( msgA );
  initQM.putMessage( targetQMGrName, targetQName, msgObj, null, 0);

  ...
  /* application on recipient QueueManager: */
  /* use MQeMTrustAttribute to recover the message from. */
  /* pre-registered initiator 'Bruce1' and recipient 'Bruce8' */
  /* assume recipient's QueueManager recipQM started */
  ...
  /* setup MQeMTrustAttribute */
  MQeMARSCryptor mars = new MQeMARSCryptor( );
  MQeMTrustAttribute msgA
    = new MQeMTrustAttribute(null, mars, null);

  /* setup recipient's Private Registry */
  String EntityName = "Bruce8";
  String EntityPIN = "12345678";
  Object KeyRingPassword = "It_is_a_secret";

  /* instantiate and activate recipient's Private Registry */
  MQePrivateRegistry recipreg = new MQePrivateRegistry( );
  recipreg.activate( EntityName, "./MQeNode_PrivateRegistry",
                  EntityPIN, KeyRingPassword, null, null );
  /* set MTrustAttribute PrivateRegistry = recipreg */
  msgA.setPrivateRegistry( recipreg );
  /* instantiate and activate Public Registry which has */
  /* (or gets) MiniCert of originator (Bruce1) */

```

MQeMTrustAttribute

```
MQePublicRegistry pr          = new MQePublicRegistry( );
pr.activate( "MQeNode_PublicRegistry", ".//" );
/* set MTrustAttribute's PublicRegistry & HomeServer          */
msgA.setPublicRegistry( pr);
msgA.setHomeServer( MyHomeServer + ":8081" );
/* use MQeMTrustAttribute with GetMessage to recover msg      */
MQeMsgObject MsgObj = SvrQM.getMessage( TargetQMgrName,
    TargetQName,null, msgA, 0 );
trace("i: Data restored from MTrustAttr protected Msg ="
    + byteToAscii(MsgObj.getArrayOfByte("TestData" ) ) );
}
```

Related functions

- MQePrivateRegistry
- MQePublicRegistry

MQeMTrustAttribute decodeData

Syntax

```
public byte[] decodeData( MQeChannel channel,
    byte data[],
    int offset,
    int count ) throws Exception
```

Description

Is called to decode (decrypt and/or decompress) the bytes referenced by **data**, **offset** and for length **count**.

Note: This method is intended for internal use and is not normally called by applications.

Parameters

channel	<i>null</i> , not used
data	An object reference to a byte array containing the data to be decoded
offset	An integer index specifying the start byte in the data array
count	An integer count of the number of bytes to decode

Return Values

Decoded data

Exceptions

MQeException	Except_S_RegistryNotAvailable "intended recipient's PrivateRegistry not available"
	Except_S_MiniCertNotAvailable "cannot recover data, sender's MiniCert not available"
	Except_S_RegistryNotAvailable "cannot recover data target(recipient) PrivateRegistry not available"
	Except_S_BadIntegrity, "validating data from < Sender> data tampering detected"
	Except_S_InvalidSignature, "validating data from < Sender > bad signature"

MQeMTrustAttribute encodeData

Syntax

```
public byte[] encodeData( MQeChannel channel,
                        byte data[],
                        int offset,
                        int count ) throws Exception
```

Description

Encodes (encrypts and/or compresses) the bytes referenced by **data**, **offset** and for length **count**.

Note: This method is intended for internal use and is not normally called by applications.

Parameters

channel	<i>null</i> , not used
data	An object reference to a byte array containing the data to be encoded
offset	An integer index specifying the start byte in the data array
count	An integer count of the number of bytes to encode

Return Values

none

Exceptions

MQeException	Except_S_MiniCertNotAvailable, "cannot protect data, target Mini Certificate not available"
	Except_S_RegistryNotAvailable "sender's PrivateRegistry not available"

MQeMTrustAttribute setHomeServer

Syntax

```
public void setHomeServer( String homeServerAddrPort)
                        throws Exception
```

Description

Called to set an MQeMTrustAttribute's *home server* address. When used to protect a message, **encodeData** attempts to get the intended recipient's Mini-Certificate recipient from its active public registry. If not found but the *home server* address is set, it requests the Mini-Certificate from the *home server*, and saves it for subsequent use in the active public registry. When used to recover a message, **decodeData** attempts to get the initiator's Mini-Certificate from its active public registry. If not found but the *home server* address is set, it requests the Mini-Certificate from the *home server* and saves it for subsequent use in its active public registry.

Parameters

homeServerAddrPort	The address of another MQeNode (for example HomeServer) with a public registry containing a larger set of authenticatable entities' Mini-Certificates. The format used is <i>tcpname:port</i> , or <i>tcpaddress:port</i>
---------------------------	---

MQeMTrustAttribute

Return Values

none

Exceptions

MQeException

Except_NotAllowed, "illegal
SetPublicRegistry"

MQeMTrustAttribute setPrivateRegistry

Syntax

```
public void setPrivateRegistry( MQePrivateRegistry privreg)  
    throws Exception
```

Description

Called to set an MQeMTrustAttribute's active private registry. When used to protect a message this is the private registry of the sender and when recovering a message this is the private registry of the recipient.

Parameters

privreg

The activated **MQePrivateRegistry** of the sender or recipient authenticatable entity

Return Values

none

Exceptions

MQeException

Except_NotAllowed, "illegal
SetPrivateRegistry"

MQeMTrustAttribute setPublicRegistry

Syntax

```
public void setPublicRegistry( MQePublicRegistry pubreg)  
    throws Exception
```

Description

Called to set an MQeMTrustAttribute's active public registry. When used to protect a message this is a public registry that has (or gets) the Mini-Certificate of the intended recipient, and when recovering a message this is a public registry that has (or gets) the Mini-Certificate of the sender.

Parameters

pubreg

An activated **MQePublicRegistry** containing the Mini-Certificate of the intended recipient, when used to protect, or the Mini-Certificate of the sender, when used to recover.

Return Values

none

Exceptions

MQeException

Except_NotAllowed, "illegal
SetPublicRegistry"

MQeRC4Cryptor

Note: This class is only available in the high security version of MQSeries Everyplace Version 1.1.

This class is used to create a RC4 cryptor object that, when used by an attribute object, provides the attribute object with the mechanisms to perform RC4 encryption. Attribute objects are associated with channel and **MQeFields** objects.

Package **com.ibm.mqe.attributes**

This class is a descendant of **MQeCryptor**

MQeRC4Cryptor

Syntax

```
public MQeRC4Cryptor( )
```

Description

Constructs an MQeRC4Cryptor object

Parameters

None

Return values

none

Exceptions

MQeException	Except_S_Cipher, "cipRC4, wrong cipher or key"
---------------------	--

Example

```
try
{
    MQeRC4Cryptor rc4    = new MQeRC4Cryptor();
    MQeAttribute rc4A    = new MQeAttribute(null, rc4, null);
    ...
}
catch ( Exception e )
{
    ...
}
```

Related functions

- **MQeCryptor**
- **MQeAttribute**
- **MQeLocalSecure**
- **MQeMAttribute**
- **MQeMTrustAttribute**

MQeRC6Cryptor

Note: This class is only available in the high security version of MQSeries Everyplace Version 1.1.

This class is used to create a RC6 cryptor object that, when used by an attribute object, provides the attribute object with the mechanisms to perform RC6 encryption. Attribute objects are associated with channel and **MQeFields** objects.

Package **com.ibm.mqe.attributes**

This class is a descendant of **MQeCryptor**

MQeRC6Cryptor

Syntax

```
public MQeRC6Cryptor( )
```

Description

Constructs an MQeRC6Cryptor object

Parameters

none

Return values

none

Exceptions

MQeException	Except_S_Cipher, "cipRC6, wrong cipher or key"
---------------------	--

Example

```
try
{
    MQeRC6Cryptor rc6 = new MQeRC6Cryptor();
    MQeAttribute rc6A = new MQeAttribute(null, rc6, null);
    ...
}
catch ( Exception e )
{
    ...
}
```

Related functions

- **MQeCryptor**
- **MQeAttribute**
- **MQeLocalSecure**
- **MQeMAttribute**
- **MQeMTrustAttribute**

MQeRleCompressor

This class is used to create an Rle compressor object that, when used by an attribute object, provides the attribute object with the mechanisms to perform Rle compression. Attribute objects are associated with channel and **MQeFields** objects.

Package **com.ibm.mqe.attributes**

This class is a descendant of **MQeCompressor**

MQeRleCompressor

Syntax

```
public MQeRleCompressor( )
```

Description

Constructs an MQeRleCompressor object

Parameters

none

Return values

none

Exceptions

none

Example

```
try
{
    MQeRleCompressor rle = new MQeRleCompressor();
    MQeAttribute rleA    = new MQeAttribute(null, null, rle);
    ...
}
catch ( Exception e )
{
    ...
}
```

Related functions

- **MQeCompressor**
- **MQeAttribute**
- **MQeLocalSecure**
- **MQeMAttribute**
- **MQeMTrustAttribute**

MQeWTLSCertAuthenticator

Note: This class is only available in the high security version of MQSeries Everyplace Version 1.0.

This class is used to create a `WTLSCertAuthenticator` object that, when used by an attribute object, provides the attribute object with the mechanisms to perform Mini-Certificate based mutual authentication. This applies to **Attribute** objects associated with channel objects.

Package **com.ibm.mqe.attributes**

This class is a descendant of **MQeAuthenticator**

MQeWTLSCertAuthenticator

Syntax

```
public MQeWTLSCertAuthenticator( )
```

Description

Constructs an `MQeWTLSCertAuthenticator` object

Parameters

none

Return values

none

Exceptions

none

Example

```
try
{
    MQeWTLSCertAuthenticator wtls = new MQeWTLSCertAuthenticator( );
    MQeDESCryptor des             = new MQeDESCryptor( );
    MQeAttribute wtlsA            = new MQeAttribute(wtls, des, null);
    ...
}
catch ( Exception e )
{
    ...
}
```

Related functions

- **MQeAuthenticator**
- **MQeAttribute**

MQeXORCryptor

This class is used to create an XOR cryptor object that, when used by an attribute object, provides the attribute object with the mechanisms to perform XOR encoding. Attribute objects are associated with channel and **MQeFields** objects.

Package **com.ibm.mqe.attributes**

This class is a descendant of **MQeCryptor**

Constructor summary

Constructor	Purpose
MQeXORCryptor	Constructs an MQeXORCryptor object

Method summary

Method	Purpose
setDecryptKey	Explicitly sets the cryptor's decrypt key
setEncryptKey	Explicitly sets the cryptor's encrypt key

MQeXORCryptor

Syntax

```
public MQeXORCryptor( )
```

Description

Constructs an MQeXORCryptor object

Parameters

none

Return values

none

Exceptions

none

Example

```
try
{
    MQeXorCryptor xor = new MQeXorCryptor( );
    xor.setEncryptKey ( asciiToByte("It_is_a_secret" ) );
    NTAuthenticator nt = new NTAuthenticator( );
    String inData      = "0123456789abcdef...";
    trace("i: TestXOR, indata = " + inData);
    MQeFields tempf    = new MQeFields();
    tempf.putAscii( "testdata", inData);
    MQeAttribute attr1 = new MQeAttribute( );
    attr1.activate( null, nt, xor, null );
    tempf.setAttribute ( attr1 );
    byte[] temp = tempf.dump();

    //

    MQeFields tempf2 = new MQeFields();
    MQeXorCryptor xor2 = new MQeXorCryptor( );
    xor2.setDecryptKey ( asciiToByte("It_is_a_secret" ) );
    NTAuthenticator nt2 = new NTAuthenticator( );
```

MQeXORCryptor

```
MQeAttribute attr2 = new MQeAttribute( );
attr2.activate( null, nt2, xor2, null );
tempf2.setAttribute ( attr2 );
tempf2.restore( temp );
trace("i: TestXORSecure, outdata = " + tempf2.getAscii("testdata"));
}
catch ( Exception e )
{
//
}
```

Related functions

- MQeCryptor
- MQeAttribute

MQeXORCryptor setDecryptKey

Syntax

```
Public void setDecryptKey ( Object newKey) throws Exception
```

Description

Explicitly sets the cryptor's decrypt key

Parameters

newKey byte[] seed from which the cryptor's decrypt key is derived.

Return values

none

Exceptions

none

MQeXORCryptor setEncryptKey

Syntax

```
Public void setEncryptKey ( Object newKey) throws Exception
```

Description

Explicitly sets the cryptor's encrypt key

Parameters

newKey byte[] seed from which the cryptor's encrypt key is derived.

Return values

none

Exceptions

none

Chapter 5. Classes in com.ibm.mqe.registry

This section contains detailed information about the following MQSeries Everyplace classes:

Table 15. Classes in package com.ibm.mqe.registry

Class name	Purpose
MQePrivateRegistry	Creates a private registry object that provides controlled access to a set of private and public objects
MQePrivateRegistryConfigure	Used to configure a private registry
MQePublicRegistry	Creates a public registry object that provides controlled access to a set of private and public objects

MQePrivateRegistry

This class is used to create an MQePrivateRegistry object. MQePrivateRegistry class is a descendent of **MQeRegistry** and provides controlled access to a set of private and public objects (for example certificates). MQePrivateRegistry objects also support digital signing and decryption services which can use the registry's private objects (for example an authenticatable entity's private key) internally, so they do not leave the *private registry*.

Package com.ibm.mqe.registry

This class is a descendant of **MQeRegistry**

Constructor summary

Constructor	Purpose
MQePrivateRegistry	Constructs an MQePrivateRegistry object

Method summary

Method	Purpose
activate	Opens and activates the MQePrivateRegistry instance.
deleteCertificate	Deletes the certificate owner's Mini-Certificate
getCertificate	Returns the certificate owner's Mini-Certificate
getRegistryName	Gets the private registry's authenticatable entity name.
resetPIN	Resets the PIN that controls private access.
setTargetRegistryName	Sets the name of the intended recipient (authenticatable entity) private registry

MQePrivateRegistry

Syntax

```
public MQePrivateRegistry( )
```

Description

Constructs an MQePrivateRegistry object

Parameters

none

Return values

none

Exceptions

none

Related functions

- MQePublicRegistry

MQePrivateRegistry activate

Syntax

```
Public void activate (String entityName,  
String dirName,  
String pin,
```



```
Object keyRingPassword,
Object certReqPIN,
Object caIPAddrPort ) throws Exception
```

Description

If a private registry with this **entityName** exists, **activate** attempts to open the private registry using the given **pin**. If it does not exist, **activate** creates and opens a new private registry and makes it accessible with the given **pin**.

If a non-null Mini-Certificate server address (**caIPAddrPort**) is provided, **activate** searches the private registry to discover if the owner is already registered (already has its own Mini-Certificate). If it is not registered (no Mini-Certificate), **activate** executes autoregistration. This autoregisters the **entityName**, performing the following tasks:

- Generates a new RSA key pair for the owning **entityName**
- Saves the private key(CRTKey) in the private registry after protecting using a derivative of the given **keyRingPassword**
- Packages the public key in a **newCertificateRequest** to the Mini-Certificate server address given, identifying the request with the **entityName** and the given (pre-allocated) Mini-Certificate request pin (**certReqPIN**)
- Saves the issued Mini-Certificate in the private registry then sends a **getCertificate** request to get the Mini-Certificate server's (own) Mini-Certificate, and saves it in the private registry

Parameters

entityName	PrivateRegistry owner EntityName
dirName	Path to PrivateRegistry
pin	Number, password or passphrase to be used to open the private registry
keyRingPassword	String password or passphrase used to protect the entity's private key
certReqPIN	String with one-time-use <i>Certificate Request Number</i> preallocated for the entity by the Mini-Certificate server administrator to enable it to autoregister
caIPAddrPort	String with the TCP address and port of the solution's Mini-Certificate server, for example <code>aname.hursley.ibm.com:8081</code>

Return Values

none

Exceptions

MQException	Except_PrivateReg_BadPIN, "Activating_EntityName_PrivateRegistry"
	Except_PrivateReg_ActivateFailed
	Except_PrivateReg_ActivateFailed, "Registration exception "

Example

MQePrivateRegistry

```
class MySampleClass extends MQe
{
  try
  {
    /* setup Private Registry activate parameters */
    String entityName      = "Bruce";
    String dirName         = ".//" + EntityName;
    String entityPIN       = "12345678";
    Object keyRingPassword = "It_is_a_secret";
    Object certReqPIN      = "12345678";
    Object caIPAddrPort    = "aname.hursley.ibm.com:8081";
    /* instantiate and activate a Private Registry... */
    MQePrivateRegistry preg = new MQePrivateRegistry( );
    /* instantiate and activate the Private Registry */
    preg.Activate( entityName, /* name of entity owning privreg */
                  dirName,    /* params to open file regsess'n */
                  entityPIN,  /* Private Registry access PIN */
                  keyRingPassword, /* pwd/phrase protecting CRTKey */
                  certReqPIN, /* prereg MiniCertSvr certreqPIN */
                  caIPAddrPort); /* trusted MiniCertSvr addr:port */
  }
  catch ( Exception e )
  {
  }
}
```

Related functions
MqeLocalSecure

MQePrivateRegistry deleteCertificate

Syntax

```
Public MQeFields deleteCertificate( String certificateOwner )
                                     throws MQeException
```

Description

Deletes the certificate owner's Mini-Certificate.

Parameters

certificateOwner
Private registry owner's name

Return Values

none

Exceptions

MQeException
Except_Reg_DoesNotExist, "Entry does not exist"
Except_Reg_DeleteFailed, "Error deleting entry"

Related functions
getCertificate

MQePrivateRegistry getCertificate

Syntax

```
Public MQeFields getCertificate( String certificateOwner )
                                     throws MQeException
```

Description

Returns the certificate owner's Mini-Certificate.

Parameters

certificateOwner
Private registry owner's name

Return Values
Mini-Certificate

Exceptions
MQeException
Except_Reg_ReadFailed, "Error reading entry"

Related functions
deleteCertificate

MQePrivateRegistry getRegistryName

Syntax
Public String getRegistryName()

Description
Returns the owning entity name

Parameters
none

Return Values
Owning entity name

Exceptions
none

MQePrivateRegistry resetPIN

Syntax
Public void resetPIN(String currentPIN,
String newPIN) throws Exception

Description
Enables a valid private registry owner to change the access PIN

Parameters

currentPIN	Current valid PIN (password or passphrase) for private registry
newPIN	New PIN (password or passphrase)

Return Values
none

Exceptions
MQeException
Except_PrivateReg_BadPIN , "PIN not reset, bad current PIN provided"

MQePrivateRegistry setTargetRegistryName

Syntax
Public void setTargetRegistryName(String registryName)

Description
Adds the name of the intended recipient's private registry.

Parameters

MQePrivateRegistry

registryName Current valid PIN (password or passphrase) for private registry

newPIN recipient's Private Registry name

Return Values

none

Exceptions

none

Examples

See "MQeMTrustAttribute" on page 244

Related functions

- **MQeMTrustAttribute**

MQePrivateRegistryConfigure

This class is used to configure a Private Registry. The class is used to get new credentials (private and public certificates) for the registry.

Package **com.ibm.mqe.registry**

This class is a descendant of **MQeRegistry**

Constructor summary

Constructor	Purpose
MQePrivateRegistryConfigure	Instantiates the registry configuration object

Method summary

Method	Purpose
activate	Initializes the class and opens the registry
close	Closes the registry and tidies up
credentialsExist	Checks whether credentials already exist for the registry
getCredentials	Obtains new credentials for the registry
isPrivate	Checks whether the registry is a private registry

MQePrivateRegistryConfigure

Syntax

1. `public MQePrivateRegistryConfigure()`
2. `public MQePrivateRegistryConfigure(String name, MQeFields parms, String PIN) throws Exception`

Description

The constructor instantiates the registry configuration object, there are two versions.

1. This is an empty constructor that is designed for dynamic loading and must be followed by a call to **activate()**
2. This version saves the name and opens the registry; it is equivalent to the empty constructor followed by **activate()**

Parameters

name The name associated with this registry

regParams An **MQeFields** object containing the initialization parameters for the registry:

MQeRegistry.LocalRegType (ascii)

This determines the type of the registry being opened. Because this should be a private registry, this parameter should be set to `com.ibm.mqe.registry.MQePrivateSession`, or an equivalent alias.

MQeRegistry.DirName (ascii)

The name of the directory holding the registry files

MQePrivateRegistryConfigur

MQeRegistry.PIN (ascii)

The PIN for the private registry. This is used if the **regPIN** parameter on **activate()** is *null*.

MQeRegistry.KeyRingPassword (ascii)

The password or passphrase used to protect the registry's private key.

MQeRegistry.Separator (ascii)

The character to be used as a separator between the components of an entry name, for example <QueueManager><Separator><Queue>.

This is specified as a string but it should contain a single character, if it contains more than one only the first character is used.

The same separator character should be used every time a registry is opened, it should not be changed once a registry is in use and contains entries.

If this value is not specified it defaults to "+".

regPIN

The PIN required to open the registry. If this is *null*, the PIN is taken from the **regParams** parameter.

Return values

none

Exceptions

Exception

Thrown if there is a problem opening the registry

Example

```
MQePrivateRegistryConfigure regConfig1;  
regConfig1 = new MQePrivateRegistryConfigure();  
  
try  
{  
    MQePrivateRegistryConfigure regConfig2;  
    MQeFields parms = new MQeFields();  
    parms.putUnicode(MQeRegistry.DirName, "Registry_Dir");  
    ...  
    regConfig2 = new MQePrivateRegistryConfigure("Reg2", parms, null);  
}  
catch (Exception e)  
{ ... }
```

MQePrivateRegistryConfigure activate

Syntax

```
public void activate( String name,  
                    MQeFields regParams,  
                    String regPIN ) throws Exception
```

Description

This saves the registry name and opens the registry. If the **regPIN** parameter is not *null* it is used to open the registry, otherwise the registry's PIN is obtained from the **regParams** parameter.

Parameters

MQePrivateRegistryConfigur

name	The name associated with this registry
regParams	An MQeFields object containing the initialization parameters for the registry. MQeRegistry.LocalRegType (ascii) This determines the type of the registry being opened. Because this should be a private registry, this parameter should be set to <code>com.ibm.mqe.registry.MQePrivateSession</code> , or an equivalent alias. MQeRegistry.DirName (ascii) The name of the directory holding the registry files MQeRegistry.PIN (ascii) The PIN for the private registry. This will be used if the regPIN parameter is <i>null</i> . MQeRegistry.KeyRingPassword (ascii) The password or passphrase used to protect the registry's private key. MQeRegistry.Separator (ascii) The character to be used as a separator between the components of an entry name, for example <code><QueueManager><Separator><Queue></code> . This is specified as a string but it should contain a single character, if it contains more than one only the first character is used. The same separator character should be used every time a registry is opened, it should not be changed once a registry is in use and contains entries. If this value is not specified it defaults to "+".
regPIN	The PIN required to open the registry. If this is <i>null</i> , the PIN is taken from the regParams parameter.

Return Values

none

Exceptions

MQeException	Is thrown if there is a problem opening the registry.
Exception	Is thrown if there are any other problems

Example

```
try
{
    MQePrivateRegistryConfigure regConfig;
    MQeFields parms = new MQeFields();
    parms.putUnicode(MQeRegistry.DirName, "Registry_Dir");
    ...
    regConfig = new MQePrivateRegistryConfigure();
}
```

MQePrivateRegistryConfigur

```
    regConfig.activate("Reg", parms, null);
  }
  catch (Exception e)
  { ... }
```

Related functions

MQeLocalSecure

MQePrivateRegistryConfigure close

Syntax

```
public void close()
```

Description

This closes the configuration object and the associated registry. An attempt to use the object after it has been closed results in an exception.

Parameters

none

Return Values

none

Exceptions

none

Example

```
try
{
  MQePrivateRegistryConfigure regConfig;
  MQeFields parms = new MQeFields();
  parms.putUnicode(MQeRegistry.DirName, "Registry_Dir");
  ...
  regConfig = new MQePrivateRegistryConfigure("Reg", parms, null);
  if ( regConfig.credentialsExist() )
  {
    ...
  }
  regConfig.close();
}
catch (Exception e)
{ ... }
```

MQePrivateRegistryConfigure credentialsExist

Syntax

```
public boolean credentialsExist ( ) throws MQeException
```

Description

This method checks whether the registry already contains credentials.

Parameters

none

Return Values

true	If the registry already contains credentials
false	If the registry does not contain credentials

Exceptions

MQeException	Is thrown if the class has not been activated
---------------------	---

Example


```

try
{
    MQePrivateRegistryConfigure regConfig;
    MQeFields parms = new MQeFields();
    parms.putUnicode(MQeRegistry.DirName, "Registry_Dir");
    ...
    regConfig = new MQePrivateRegistryConfigure("Reg", parms, null);
    if ( regConfig.credentialsExist() )
    {
        ...
    }
}
catch (Exception e)
{ ... }

```

MQePrivateRegistryConfigure getCredentials

Syntax

```

public void getCredentials( MQeFields regParams,
                           String regPIN,
                           String minCertServer,
                           String miniCertPIN,
                           String renamePrefix ) throws Exception

```

Description

This creates new credentials for the registry.

If the registry already contains credentials, they are renamed using the **renamePrefix**. If the rename fails, for example because the new name already exists in the registry, an exception is thrown and new credentials are not obtained. If an error occurs after the credentials have been renamed, they are changed back to their original names before **getCredentials()** returns.

This method calls the Mini-Certificate server and can take some time to complete.

Parameters

regParams An **MQeFields** object containing the initialization parameters for the registry.

MQeRegistry.LocalRegType (ascii)

This determines the type of the registry being opened. Because this should be a private registry, this parameter should be set to `com.ibm.mqe.registry.MQePrivateSession`, or an equivalent alias.

MQeRegistry.DirName (ascii)

The name of the directory holding the registry files

MQeRegistry.PIN (ascii)

The PIN for the private registry. This is used if the **regPIN** parameter on **activate()** is *null*.

MQeRegistry.KeyRingPassword (ascii)

The password or passphrase used to protect the registry's private key.

MQeRegistry.Separator (ascii)

MQePrivateRegistryConfigur

The character to be used as a separator between the components of an entry name, for example <QueueManager><Separator><Queue>.

This is specified as a string but it should contain a single character, if it contains more than one only the first character is used.

The same separator character should be used every time a registry is opened, it should not be changed once a registry is in use and contains entries.

If this value is not specified it defaults to "+".

regPIN	The PIN required to open the registry. If this is <i>null</i> , the PIN is taken from the regParams parameter.
minCertServer	The TCP address and port number of a Mini-Certificate server
miniCertPIN	The <i>Certificate Request Number</i> pre-allocated by the Mini-Certificate administrator to allow the registry to obtain its credentials
renamePrefix	A prefix used to rename the existing credentials, if there are any

Return Values

none

Exceptions

MQeException Is thrown if the class has not been activated, if it is not a Private Registry, if the rename fails, or if there is an error obtaining the credentials (e.g. contacting the Issuing Server).

Exception Is thrown for other errors

Example

```
try
{
    MQePrivateRegistryConfigure regConfig;
    MQeFields parms = new MQeFields();
    parms.putUnicode(MQeRegistry.DirName, "Registry_Dir");
    ...
    regConfig = new MQePrivateRegistryConfigure("Reg", parms, null);
    if ( regConfig.isPrivate() )
    {
        String renamePref = Long.toString(new Date().getTime()) + "_";
        regConfig.getCredentials( parms,
            "MYpin 123",
            "certServer.hursley.ibm.com:8081",
            "12345678",
            renamePref );
    }
}
catch (Exception e)
{ ... }
```

MQePrivateRegistryConfigure isPrivate**Syntax**

```
public boolean isPrivate( ) throws MQeException
```

Description

This method checks whether the registry that has been opened is a private registry.

Parameters

none

Return Values

true	If it is a private registry
false	If it is not a private registry

Exceptions

MQeException	Is thrown if the class has not been activated
---------------------	---

Example

```
try
{
    MQePrivateRegistryConfigure regConfig;
    MQeFields parms = new MQeFields();
    parms.putUnicode(MQeRegistry.DirName, "Registry_Dir");
    ...
    regConfig = new MQePrivateRegistryConfigure("Reg", parms, null);
    if ( regConfig.isPrivate() )
    {
        ...
    }
}
catch (Exception e)
{ ... }
```

MQePublicRegistry

This class is used to create a MQePublicRegistry object.

Package **com.ibm.mqe.registry**

This class is a descendant of **MQeRegistry**

Constructor summary

Constructor	Purpose
MQePublicRegistry	Constructs an MQePublicRegistry object

Method summary

Method	Purpose
activate	Opens and activates the MQePublicRegistry instance.
deleteCertificate	Deletes the certificate owner's Mini-Certificate
getCertificate	Returns the certificate owner's Mini-Certificate
putCertificate	Adds the certificate owner's Mini-Certificate to the public registry
requestCertificate	Requests a Mini-Certificate from the public registry of another MQeNode
shareCertificate	Replicates the certificate owner's Mini-Certificate to a public registry on another MQeNode

MQePublicRegistry

Syntax

```
public MQePublicRegistry( )
```

Description

Constructs an MQePublicRegistry object

Parameters

none

Return values

none

Exceptions

none

Related functions

- **MQePrivateRegistry**

MQePublicRegistry activate

Syntax

```
Public void activate (String name,  
String dirName) throws Exception
```

Description

If a public registry with this entity name exists, **activate** opens the existing public registry, if not it creates a new public registry with name **name**.

Parameters

name	Public registry name, normally MQeNode_PublicRegistry
dirName	Path to public registry

Return Values

none

Exceptions

MQeException	Except_Public_ActivateFailed, "exception reason"
---------------------	--

Example

```
class MySampleClass extends MQe
{
try
{
/* setup Public Registry activate parameters */
String name = "MQeNode_PublicRegistry";
String dirName = "./"
/* instantiate and activate Public Registry */
MQePublicRegistry pubreg = new MQePublicRegistry( );
pubreg.activate( name, dirName );
}
catch ( Exception e )
{
...
}
```

Related functions

MQePrivateRegistry

MQePublicRegistry deleteCertificate**Syntax**

```
Public void deleteCertificate( String certificateOwner )
throws MQeException
```

Description

Deletes the certificate owner's Mini-Certificate.

Parameters

certificateOwner	Mini-Certificate owner's name
-------------------------	-------------------------------

Return Values

none

Exceptions

MQeException	Except_Reg_DoesNotExist, "Entry does not exist"
	Except_Reg_DeleteFailed, "Error deleting entry"

Related functions

- getCertificate
- putCertificate

MQePublicRegistry

MQePublicRegistry getCertificate

Syntax

```
Public MQeFields getCertificate( String certificateOwner )  
                                throws MQeException
```

Description

Returns the certificate owner's Mini-Certificate.

Parameters

certificateOwner
Authenticatable entity's (Mini-Certificate owner's) name

Return Values

Mini-Certificate

Exceptions

MQeException Except_Reg_ReadFailed, "Error reading entry"

Related functions

- deleteCertificate
- putCertificate

MQePublicRegistry putCertificate

Syntax

```
Public void putCertificate( String certificateOwner,  
                           MQeFields certificate ) throws MQeException
```

Description

Adds the certificate owner's Mini-Certificate to the public registry

Parameters

certificateOwner
Authenticatable entity's (Mini-Certificate owner's) name

certificate Owner's Mini-Certificate

Return Values

none

Exceptions

MQeException Except_Reg_AlreadyExists, "Entry already exists"

 Except_Reg_AddFailed, "Error adding entry"

Related functions

- getCertificate
- deleteCertificate

MQePublicRegistry requestCertificate

Syntax

```
Public MQeFields requestCertificate( String certificateOwner,  
                                     String mqeNodeAddrPort)  
                                    throws MQeException
```

Description

Requests a Mini-Certificate from the public registry of another MQeNode and, if returned, saves it in this publicRegistry.

Parameters**certificateOwner**

Mini-Certificate owner's name

mqeNodeAddrPort

TCP address and port of *home server* or alternative MQeNode

Return Values

Mini-Certificate

Exceptions**MQeException**

Except_Reg_DoesNotExist, "Entry does not exist"
 Except_Reg_ReadFailed, "Error reading entry"
 Except_Reg_AddFailed, "Error adding entry"

Example

```
class MySampleClass extends MQe
{
  try
  {
    /* setup RequestCertificate parameters */
    String homeServerAddrPort = "homeServer.hursley.ibm.com:8081";
    String entityName         = "Bruce";
    /* instantiate and activate Public Registry */
    MQePublicRegistry pubreg = new MQePublicRegistry( );
    pubreg.activate("MQeNode_PublicRegistry", ".\\");
    /* request Bruce's MiniCert from Public Reg on another MQeNode */
    MQeFields minicertf      = pubreg.getCertificate( entityName,
                                                       homeServerAddrPort);

    pubreg.close();
  }
  catch ( Exception e )
  {
    ...
  }
}
```

Related functions

shareCertificate

MQePublicRegistry shareCertificate**Syntax**

```
Public void shareCertificate( String certificateOwner,
                             MQeFields certificate,
                             String mqeNodeAddrPort) throws MQeException
```

Description

Replicates the certificate owner's Mini-Certificate to a public registry on another MQeNode.

Parameters**certificateOwner**

Mini-Certificate owner's name

certificate

Mini-Certificate

MQePublicRegistry

mqeNodeAddrPort

TCP address and port of *home server* or alternative MQeNode

Return Values

none

Exceptions

MQeException

Except_Reg_DoesNotExist, "Entry does not exist"

Except_Reg_ReadFailed, "Error reading entry"

Except_Reg_AddFailed, "Error adding entry"

Example

```
{
  try
  {
    /* instantiate & activate a Private Reg for Auth Entity Bruce */
    entityName          = "Bruce;
    caIPAddrPort       = "aname.hursley.ibm.com:8081";
    MQePrivateRegistry preg = new MQePrivateRegistry( );
    preg.activate( entityName, ".\MQeNode_PrivateRegistry",
                    "12345678", "It is a secret", "12345678", caIPAddrPort);
    /* instantiate and activate Public Reg & save Bruce's MiniCert */
    MQePublicRegistry pubreg = new MQePublicRegistry( );
    pubreg.activate("MQeNode_PublicRegistry",
                  ".\MQeNode_PublicRegistry" );
    pubreg.putCertificate( entityName,
                          preg.getCertificate( entityName ) );
    /* share Bruce's MiniCert with Public Reg on another MQeNode */
    String homeServerAddrPort = "homeServer.hursley.ibm.com:8081";
    pubreg.shareCertificate( entityName,
                            preg.getCertificate( entityName ), homeServerAddrPort);
    preg.close();
    pubreg.close();
  }
  catch ( Exception e )
  {
  }
}
```

Related functions

requestCertificate

Chapter 6. Classes in com.ibm.mqe.server

This section contains detailed information about the following MQSeries Everyplace classes:

Table 16. Classes in package com.ibm.mqe.server

Class name	Purpose
**MQeMiniCertIssuanceInterface	Used to define the way in which instances of MQeMiniCertificateServerGUI manages new Mini-Certificate issuance

Note: Classes marked ** are available only in the high security version of MQSeries Everyplace Version 1.1.

MQeMiniCertIssuanceInterface

Note: This class is only available in the high security version of MQSeries Everyplace Version 1.1.

Implementations of this interface are used to define the way in which instances of MQeMiniCertificateServerGUI manage new Mini-Certificate issuance. The default implementation, **MQeMiniCertIssuanceManager** uses an **MQeMiniCertificateRegistry** as the repository for definitions of a valid set of authenticatable entities that can request Mini-Certificates. It is recognized that MQSeries Everyplace solutions may want to use different repositories for this data, for example other registry or database services.

Package **com.ibm.mqe.server**

Constants

This class provides the following constants:

```
public final static int     IssMgr_OK
public final static int     IssMgr_Error
```

Method summary

Method	Purpose
addAuthenticatableEntity	Add the name and one-time-use <i>certificate request PIN</i> for a valid MQSeries Everyplace solution authenticatable entity
addEntityRegisteredAddress	Add the registered address for a valid MQSeries Everyplace solution authenticatable entity
authoriseMiniCertRequest	Authorize a new Mini-Certificate request.
deleteAuthenticatableEntity	Delete the name and one-time-use <i>certificate request PIN</i> of a valid MQSeries Everyplace solution authenticatable entity
deleteEntityRegisteredAddress	Delete the registered address of a valid MQSeries Everyplace solution authenticatable entity
readAuthenticatableEntity	Read the name and one-time-use <i>certificate request PIN</i> of a valid MQSeries Everyplace solution authenticatable entity
readEntityRegisteredAddress	Read the registered address of a valid MQSeries Everyplace Solution authenticatable entity
setRegistry	Set the reference to the MQeRegistry repository instance
updateAuthenticatableEntity	Update the name and one-time-use <i>certificate request PIN</i> of a valid MQSeries Everyplace solution authenticatable entity
updateEntityRegisteredAddress	Update the registered address of a valid MQSeries Everyplace solution authenticatable entity.

MQeMiniCertIssuanceInterface addAuthenticatableEntity

Syntax

```
public int addAuthenticatableEntity (String entityName,String certReqPIN )
```

Description

Add the name and one-time-use *certificate request PIN* for a valid MQSeries Everyplace solution authenticatable entity

Parameters

entityName A String used to identify the authenticatable entity's name
certReqPIN A String used to identify the authenticatable entity's one-time-use *certificate request PIN*

Return Values

- IssMgr_OK to indicate success
- IssMgr_Error to indicate failure

Exceptions

none

MQeMiniCertIssuanceInterface addEntityRegisteredAddress**Syntax**

```
public int addEntityRegisteredAddress (String entityName,
                                     MQeFields entityRegAddr )
```

Description

Add the registered address for a new authenticatable entity

Parameters

entityName A String used to identify the authenticatable entity's name
entityRegAddr An MQeFields object containing the authenticatable entity's registered address

Return Values

- IssMgr_OK to indicate success
- IssMgr_Error to indicate failure

Exceptions

none

MQeMiniCertIssuanceInterface authoriseMiniCertRequest**Syntax**

```
public int authoriseMiniCertRequest (String entityName,String certReqPIN )
```

Description

Authorize a new Mini-Certificate request

Parameters

entityName A String used to identify the authenticatable entity's name
certReqPIN A String used to identify the authenticatable entity's one-time-use *certificate request PIN*

Return Values

- IssMgr_OK to indicate success
- IssMgr_Error to indicate failure

Exceptions

none

MQeMiniCertIssuanceInterface

MQeMiniCertIssuanceInterface deleteAuthenticatableEntity

Syntax

```
public int deleteAuthenticatableEntity (String entityName)
```

Description

Delete the name and one-time-use *certificate request PIN* of a valid MQSeries Everyplace solution authenticatable entity.

Parameters

entityName A String used to identify the authenticatable entity's name

Return Values

- IssMgr_OK to indicate success
- IssMgr_Error to indicate failure

Exceptions

none

MQeMiniCertIssuanceInterface deleteEntityRegisteredAddress

Syntax

```
public int deleteEntityRegisteredAddress (String entityName)
```

Description

Delete the registered address of a valid MQSeries Everyplace solution authenticatable entity.

Parameters

entityName A String used to identify the authenticatable entity's name

Return Values

- IssMgr_OK to indicate success
- IssMgr_Error to indicate failure

Exceptions

none

MQeMiniCertIssuanceInterface readAuthenticatableEntity

Syntax

```
public int authoriseMiniCertRequest (String entityName,String certReqPIN )
```

Description

Read the name and one-time-use *certificate request PIN* of a valid MQSeries Everyplace solution authenticatable entity

Parameters

entityName A String used to identify the authenticatable entity's name

byte< > A String used to identify the authenticatable entity's one-time-use *certificate request PIN*

Return Values

A byte array containing authenticatable entity's identity, or *null*

Exceptions

none

MQeMiniCertIssuanceInterface readEntity RegisteredAddress**Syntax**

```
public MQeFields readEntityRegisteredAddress (String entityName)
```

Description

Read the registered address of a valid MQSeries Everyplace solution authenticatable entity.

Parameters

entityName A String used to identify the authenticatable entity's name

Return Values

An **MQeFields** object containing the authenticatable entity's registered address or *null*

Exceptions

none

MQeMiniCertIssuanceInterface SetRegistry**Syntax**

```
public void setRegistry (MQeRegistry registry)
```

Description

Set the reference to a MQeRegistry repository instance

Parameters

registry MQeRegistry instance

Return Values

none

Exceptions

none

MQeMiniCertIssuanceInterface updateAuthenticatableEntity**Syntax**

```
public int updateAuthenticatableEntity (String entityName,
                                       String certReqPIN )
```

Description

Update the name and one-time-use *certificate request PIN* of a valid MQSeries Everyplace solution authenticatable entity

Parameters

entityName A String used to identify the authenticatable entity's name

certReqPIN A String used to identify the authenticatable entity's one-time-use *certificate request PIN*

Return Values

- IssMgr_OK to indicate success
- IssMgr_Error to indicate failure

Exceptions

none

MQeMiniCertIssuanceInterface

MQeMiniCertIssuanceInterface updateEntityRegisteredAddress

Syntax

```
public int updateEntityRegisteredAddress (String entityName,  
                                         MQeFields entityRegAddr )
```

Description

Update the registered address of a registered address for a new authenticatable entity authenticatable entity.

Parameters

entityName A String used to identify the authenticatable entity's name

entityRegAddr An MQeFields object containing the authenticatable entity's registered address

Return Values

- IssMgr_OK to indicate success
- IssMgr_Error to indicate failure

Exceptions

none

Chapter 7. Classes in com.ibm.mqe.mqemqmessage

This section contains detailed information about the following MQSeries Everyplace classes:

Table 17. Classes in package com.ibm.mqe.mqemqmessage

Class name	Purpose
MQeMQMsgObject	Used to represent an MQSeries style message object within MQSeries Everyplace

MQeMQMsgObject class

This section describes the Java class used to represent an MQSeries style message object within MQSeries Everyplace. It can be used to create and read MQSeries style message objects.

The class has `getxxx()` and `setxxx()` methods for all the MQSeries message header fields. For efficiency however, only fields that have been set to a non-default value are actually contained in the message object .

Package **com.ibm.mqe.mqemqmessage**

This class is a descendant of **MQeMsgObject**

Constructor summary

Constructor	Purpose
MQeMQMsgObject	Creates a new MQeMQMsgObject

Method summary

Method	Purpose
<code>dumpAllToString</code>	Dumps all the field values from the message to a string
<code>dumpToString</code>	Dumps the field values in the message object to a string
<code>equals</code>	Compares to byte arrays for equality
<code>getAccountingToken</code>	Gets the value of the Accounting Token from the message header
<code>getApplicationIdData</code>	Gets the Application Id Data from the message header
<code>getApplicationOriginData</code>	GetApplicationOriginData
<code>getBackoutCount</code>	Gets the Backout Count from the message header
<code>getCharacterSet</code>	Gets the coded Character Set Identifier from the message header
<code>getCorrelationId</code>	Gets the Correlation Id from the message header
<code>getdata</code>	Gets the message Data
<code>getEncoding</code>	Gets the Encoding value from the message header
<code>getExpiry</code>	Gets the Expiry value from the message header
<code>getFeedback</code>	Gets the Feedback value from the message header
<code>getFormat</code>	Gets the Format value from the message header
<code>getGroupId</code>	Gets the value of the Group Id from the message header
<code>getMessageFlags</code>	Gets the value of the Message Flags from the message header
<code>getMessageId</code>	Gets the Message Id from the message header
<code>getMessageSequenceNumber</code>	Gets the Message Sequence Number from the message header
<code>getMessageType</code>	Gets the Message Type from the message header
<code>getOffset</code>	Gets the value of the Offset from the message header

Method	Purpose
getOriginalLength	Gets the Original Length from the message header
getPersistence	Gets the Persistence value from the message header
getPriority	Gets the Priority from the message header
getPutApplicationName	Gets the Put Application Name from the message header
getPutApplicationType	Gets the Put Application Type from the message header
getPutDateTime	Gets the Put Date and Time from the message header
getReplyToQueueManagerName	Gets the ReplyTo Queue Manager Name from the message header
getReplyToQueueName	Gets the ReplyTo Queue Name from the message header
getReport	Gets the Report value from the message header
getUserId	Gets the User Id from the message header
setAccountingToken	Sets the value of the Accounting Token in the message header
setApplicationIdData	Sets the Application Id Data in the message header
setApplicationOriginData	Sets the Application Origin Data in the message header
setBackoutCount	Sets the Backout Count in the message header
setCharacterSet	Sets the coded Character Set Identifier in the message header
setCorrelationId	Sets the Correlation Id in the message header
setdata	Sets the message Data
setEncoding	Sets the Encoding value in the message header
setExpiry	Sets the Expiry value in the message header
setFeedback	Sets the Feedback value in the message header
setFormat	Sets the Format value in the message header
setGroupId	Sets the value of the Group Id in the message header
setMessageFlags	Sets the value of the Message Flags in the message header
setMessageId	Sets the Message Id in the message header
setMessageSequenceNumber	Sets the Message Sequence Number in the message header
setMessageType	Sets the Message Type in the message header
setOffset	Sets the value of the Offset in the message header
setOriginalLength	Sets the Original Length in the message header
setPersistence	Sets the Persistence value in the message header
setPriority	Sets the Priority in the message header
setPutApplicationName	Sets the Put Application Name in the message header
setPutApplicationType	Sets the Put Application Type in the message header
setPutDateTime	Sets the Put Date and Time in the message header
setReplyToQueueManagerName	Sets the ReplyTo Queue Manager Name in the message header
setReplyToQueueName	Sets the ReplyTo Queue Name in the message header

MQeMQMsgObject

Method	Purpose
setReport	Sets the Report value in the message header
setUserId	Sets the User Id in the message header

MQeMQMsgObject

Syntax

```
public MQeMQMsgObject( ) throws Exception
```

Description

This creates a new MQeMQMsgObject

Parameters

none

Return values

none

Exceptions

java.lang.Exception

Propagated from the super-class constructor, MQeMsgObject()

Example

```
...
MQeMQMsgObject MQMsg = new MQeMQMsgObject();
...
}
catch (Exception e)
{
...
}
```

MQeMQMsgObject dumpAllToString

Syntax

```
public String dumpAllToString()
```

Description

This method dumps all the header fields from the MQSeries style message with their corresponding field values to a string, together with the value of the data field. It is useful when debugging.

This method dumps all the header fields to a string. The dumpToString() method dumps only the fields that have been set to non-default values.

Parameters

none

Return values

A String containing the field names and values and the data value.

Exceptions

none

Example

```
if (msgObj instanceof MQeMQMsgObject)
{
System.out.println(((MQeMQMsgObject)msgObj).dumpAllToString());
}
```

MQeMQMMsgObject dumpToString

Syntax

```
public String dumpToString()
```

Description

This method dumps header fields from the &msg; style message with their corresponding field values to a string, together with the value of the data field. It is useful when debugging.

This method dumps only the fields that have been set to non-default values. The `dumpAllToString()` method dumps all the header fields to a string

Parameters

none

Return values

A String containing the field names and values and the data value.

Exceptions

none

Example

```
if (msgObj instanceof MQeMQMMsgObject)
{
    System.out.println(((MQeMQMMsgObject)msgObj).dumpToString());
}
```

MQeMQMMsgObject equals

Syntax

```
public boolean equals(byte [] b1, byte [] b2)
```

Description

This compares two byte arrays for equality. They are considered equal if they are the same length and each byte in one array is equal to the corresponding byte in the other array.

Parameters

b1 The first byte array for comparison

b2 The second byte array for comparison

Return values

'true' if the byte arrays are equal in length and content, otherwise 'false'

Exceptions

none

Example

```
byte [] correlId = ...
if (mqMsgObj.equals(mqMsgObj.getCorrelationId(), correlId) )
{
    ...
}
```

MQeMQMMsgObject getAccountingToken

Syntax

```
public byte [] getAccountingToken() throws Exception
```

MQeMQMMsgObject

Description

This method returns the value of the Accounting Token header field.

Parameters

none

Return values

A byte array containing the value of the Accounting Token.

Exceptions

java.lang.Exception if there is an error reading the value from the message object

Examples

```
try
{
    if (msgObj instanceof MQeMQMMsgObject)
    {
        MQeMQMMsgObject mqMsgObj = (MQeMQMMsgObject)msgObj;
        byte [] accountToken = mqMsgObj.getAccountingToken();
        ...
    }
}
catch (Exception e)
{
    ...
}
```

MQeMQMMsgObject getApplicationIdData

Syntax

```
MQeMQMMsgObject getApplicationOriginData
```

Description

This method returns the value of the Application Id Data header field.

Parameters

none

Return values

A string containing the value of the Application Id Data.

Exceptions

java.lang.Exception if there is an error reading the value from the message object

Example

```
try
{
    if (msgObj instanceof MQeMQMMsgObject)
    {
        MQeMQMMsgObject mqMsgObj = (MQeMQMMsgObject)msgObj;
        String appIdData = mqMsgObj.getApplicationIdData();
        α
    }
}
catch (Exception e)
{
    ...
}
```

MQeMQMMsgObject getApplicationOriginData

Syntax

```
public String getApplicationOriginData() throws Exception
```

Description

This method returns the value of the Application Origin Data header field.

Parameters

none

Return values

A string containing the value of the Application Origin Data

Exceptions

java.lang.Exception	if there is an error reading the value from the message object
----------------------------	--

Example

```
try
{
    if (msgObj instanceof MQeMQMMsgObject)
    {
        MQeMQMMsgObject mqMsgObj = (MQeMQMMsgObject)msgObj;
        String appOriginData = mqMsgObj.getApplicationOriginData();
        ...
    }
}
catch (Exception e)
{
    ...
}
```

MQeMQMMsgObject getBackoutCount

Syntax

```
public int getBackoutCount() throws Exception
```

Description

This method returns the value of the Backout Count header field.

Parameters

none

Return values

An int containing the value of the Backout Count.

Exceptions

java.lang.Exception	if there is an error reading the value from the message object
----------------------------	--

Example

```
try
{
    if (msgObj instanceof MQeMQMMsgObject)
    {
        MQeMQMMsgObject mqMsgObj = (MQeMQMMsgObject)msgObj;
        int backoutCount = mqMsgObj.getBackoutCount();
        ...
    }
}
```

MQeMQMsgObject

```
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject getCharacterSet

Syntax

```
public int getCharacterSet() throws Exception
```

Description

This method returns the value of the coded Character Set Identifier header field

Parameters

none

Return values

An int containing the value of the coded Character Set Identifier

Exceptions

java.lang.Exception	if there is an error reading the value from the message object
----------------------------	--

Example

```
try
{
    if (msgObj instanceof MQeMQMsgObject)
    {
        MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
        int charSet = mqMsgObj.getCharacterSet();
        ...
    }
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject getData

Syntax

```
public byte [] getData() throws Exception
```

Description

This method returns the message data. The application must know how to interpret the data.

Parameters

none

Return values

A byte array containing the message data

Exceptions

java.lang.Exception	if there is an error reading the value from the message object
----------------------------	--

Example

```
try
{
    if (msgObj instanceof MQeMQMsgObject)
    {
```

```

MQeMQMMsgObject mqMsgObj = (MQeMQMMsgObject)msgObj;
byte [] msgData = mqMsgObj.getData();
...
}
}
catch (Exception e)
{
...
}

```

MQeMQMMsgObject getCorrelationId

Syntax

```
public byte [] getCorrelationId() throws Exception
```

Description

This method returns the value of the Correlation Id header field.

Parameters

none

Return values

A byte array containing the value of the Correlation Id.

Exceptions

java.lang.Exception	if there is an error reading the value from the message object
----------------------------	--

Example

```

try
{
    if (msgObj instanceof MQeMQMMsgObject)
    {
        MQeMQMMsgObject mqMsgObj = (MQeMQMMsgObject)msgObj;
        byte [] correlId = mqMsgObj.getCorrelationId();
        ...
    }
}
catch (Exception e)
{
...
}

```

MQeMQMMsgObject getEncoding

Syntax

```
MQeMQMMsgObject getEncoding
```

Description

This method returns the value of the Encoding header field

Parameters

none

Return values

An int containing the Encoding value

Exceptions

java.lang.Exception	if there is an error reading the value from the message object
----------------------------	--

Example

MQeMQMsgObject

```
try
{
    if (msgObj instanceof MQeMQMsgObject)
    {
        MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
        int encode = mqMsgObj.getEncoding();
        ...
    }
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject getExpiry

Syntax

```
public int getExpiry() throws Exception
```

Description

This method returns the value of the Expiry header field. The value is in tenths of a second, as is used in MQSeries messages (it is not in milliseconds, which is used for the MQSeries Everyplace expiry time).

Parameters

none

Return values

An int containing the Expiry value.

Exceptions

<code>java.lang.Exception</code>	if there is an error reading the value from the message object
----------------------------------	--

Example

```
try
{
    if (msgObj instanceof MQeMQMsgObject)
    {
        MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
        int expiry = mqMsgObj.getExpiry();
        ...
    }
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject getFeedback

Syntax

```
MQeMQMsgObject getFeedback
```

Description

This method returns the value of the Feedback header field.

Parameters

none

Return values

An int containing the Feedback value.

Exceptions

java.lang.Exception	if there is an error reading the value from the message object
----------------------------	--

Example

```
try
{
    if (msgObj instanceof MQeMQMMsgObject)
    {
        MQeMQMMsgObject mqMsgObj = (MQeMQMMsgObject)msgObj;
        int feedback = mqMsgObj.getFeedback();
        ...
    }
}
catch (Exception e)
{
    ...
}
```

MQeMQMMsgObject getFormat**Syntax**

```
public String getFormat() throws Exception
```

Description

This method returns the value of the Format header field.

Parameters

none

Return values

A String containing the Format value

Exceptions

java.lang.Exception	if there is an error reading the value from the message object
----------------------------	--

Example

```
try
{
    if (msgObj instanceof MQeMQMMsgObject)
    {
        MQeMQMMsgObject mqMsgObj = (MQeMQMMsgObject)msgObj;
        String format = mqMsgObj.getFormat();
        ...
    }
}
catch (Exception e)
{
    ...
}
```

MQeMQMMsgObject getGroupId**Syntax**

```
public byte [] getGroupId() throws Exception
```

Description

This method returns the value of the Group Id header field

Parameters

none

MQeMQMsgObject

Return values

A byte array containing the value of the Group Id.

Exceptions

java.lang.Exception if there is an error reading the value from the message object

Example

```
try
{
    if (msgObj instanceof MQeMQMsgObject)
    {
        MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
        byte [] groupId = mqMsgObj.getGroupId();
        ...
    }
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject getMessageFlags

Syntax

```
public int getMessageFlags() throws Exception
```

Description

This method returns the value of the Message Flags header field

Parameters

none

Return values

This method returns the value of the Message Flags header field

Exceptions

java.lang.Exception if there is an error reading the value from the message object

Example

```
try
{
    if (msgObj instanceof MQeMQMsgObject)
    {
        MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
        int msgFlags = mqMsgObj.getMessageFlags();
        ...
    }
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject getMessageId

Syntax

```
public byte [] getMessageId() throws Exception
```

Description

This method returns the value of the Message Id header field.

Parameters

none

Return values

A byte array containing the value of the Message Id.

Exceptions

java.lang.Exception if there is an error reading the value from the message object

Example

```
try
{
    if (msgObj instanceof MQeMQMsgObject)
    {
        MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
        byte [] msgId = mqMsgObj.getMessageId();
        ...
    }
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject getMessageSequenceNumber**Syntax**

```
public int getMessageSequenceNumber() throws Exception
```

Description

This method returns the value of the Message Sequence Number header field.

Parameters

none

Return values

An int containing the value of the Message Sequence Number.

Exceptions

java.lang.Exception if there is an error reading the value from the message object

Example

```
try
{
    if (msgObj instanceof MQeMQMsgObject)
    {
        MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
        int msgSeqNo = mqMsgObj.getMessageSequenceNumber();
        ...
    }
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject getMessageType**Syntax**

```
public int getMessageType() throws Exception
```

MQeMQMsgObject

Description

This method returns the value of the Message Type header field.

Parameters

none

Return values

An int containing the value of the Message Type

Exceptions

java.lang.Exception if there is an error reading the value from the message object

Example

```
try
{
    if (msgObj instanceof MQeMQMsgObject)
    {
        MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
        int msgType = mqMsgObj.getMessageType();
        ...
    }
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject getOffset

Syntax

```
public int getOffset() throws Exception
```

Description

This method returns the value of the Offset header field.

Parameters

none

Return values

An int containing the Offset value

Exceptions

java.lang.Exception if there is an error reading the value from the message object

Example

```
try
{
    if (msgObj instanceof MQeMQMsgObject)
    {
        MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
        int offset = mqMsgObj.getOffset();
        ...
    }
}
catch (Exception e)
{
    ...
}
```

MQeMQMMsgObject getOriginalLength

Syntax

```
public int getOriginalLength() throws Exception
```

Description

This method returns the value of the Original Length header field

Parameters

none

Return values

An int containing the value of the Original Length

Exceptions

java.lang.Exception	if there is an error reading the value from the message object
----------------------------	--

Example

```
try
{
    if (msgObj instanceof MQeMQMMsgObject)
    {
        MQeMQMMsgObject mqMsgObj = (MQeMQMMsgObject)msgObj;
        int originalLength = mqMsgObj.getOriginalLength();
        ...
    }
}
catch (Exception e)
{
    ...
}
```

MQeMQMMsgObject getPersistence

Syntax

```
public int getPersistence() throws Exception
```

Description

This method returns the value of the Persistence header field

Parameters

none

Return values

An int containing the Persistence value.

Exceptions

java.lang.Exception	if there is an error reading the value from the message object
----------------------------	--

Example

```
try
{
    if (msgObj instanceof MQeMQMMsgObject)
    {
        MQeMQMMsgObject mqMsgObj = (MQeMQMMsgObject)msgObj;
        int persistence = mqMsgObj.getPersistence();
        ...
    }
}
```

MQeMQMsgObject

```
    catch (Exception e)
    {
        ...
    }
```

MQeMQMsgObject getPriority

Syntax

```
public int getPriority() throws Exception
```

Description

This method returns the value of the Priority header field.

Parameters

none

Return values

An int containing the Priority value

Exceptions

java.lang.Exception	if there is an error reading the value from the message object
----------------------------	--

Example

```
try
{
    if (msgObj instanceof MQeMQMsgObject)
    {
        MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
        int priority = mqMsgObj.getPriority();
        ...
    }
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject getPutApplicationName

Syntax

```
public String getPutApplicationName() throws Exception
```

Description

This method returns the value of the Put Application Name header field

Parameters

none

Return values

A String containing the value of the Put Application Name

Exceptions

java.lang.Exception	if there is an error reading the value from the message object
----------------------------	--

Example

```
try
{
    if (msgObj instanceof MQeMQMsgObject)
    {
        MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
        String putApp1Name = mqMsgObj.getPutApplicationName();
    }
}
```

```

    ...
  }
  catch (Exception e)
  {
    ...
  }

```

MQeMQMsgObject getPutApplicationType

Syntax

```
public int getPutApplicationType() throws Exception
```

Description

This method returns the value of the Put Application Type header field.

Parameters

none

Return values

An int containing the value of the Put Application Type

Exceptions

java.lang.Exception	if there is an error reading the value from the message object
----------------------------	--

Example

```

try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    int putAppType = mqMsgObj.getPutApplicationType();
    ...
  }
}
catch (Exception e)
{
  ...
}

```

MQeMQMsgObject getPutDateTime

Syntax

```
public GregorianCalendar getPutDateTime() throws Exception
```

Description

This method returns the value of the Put Date Time header field. The value is returned as a Gregorian Calendar object, for consistency with the MQSeries Java Client.

Parameters

none

Return values

A Gregorian Calendar object containing the Put Date and Time value

Exceptions

java.lang.Exception	if there is an error reading the value from the message object
----------------------------	--

Example

MQeMQMsgObject

```
try
{
    if (msgObj instanceof MQeMQMsgObject)
    {
        MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
        GregorianCalendar putDateTime = mqMsgObj.getPutDateTime();
        ...
    }
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject getReplyToQueueManagerName

Syntax

```
MQeMQMsgObject getReplyToQueueManagerName
```

Description

This method returns the value of the Reply To Queue Manager Name header field.

Parameters

none

Return values

A String containing the value of the Reply To Queue Manager Name.

Exceptions

java.lang.Exception	if there is an error reading the value from the message object
----------------------------	--

Example

```
try
{
    if (msgObj instanceof MQeMQMsgObject)
    {
        MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
        String replyToQueueMgrName = mqMsgObj.getReplyToQueueManagerName();
        ...
    }
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject getReplyToQueueName

Syntax

```
public String getReplyToQueueName() throws Exception
```

Description

This method returns the value of the Reply To Queue Name header field.

Parameters

none

Return values

A String containing the value of the Reply To Queue Name.

Exceptions

java.lang.Exception

if there is an error reading the value from the message object

Example

```

try
{
    if (msgObj instanceof MQeMQMMsgObject)
    {
        MQeMQMMsgObject mqMsgObj = (MQeMQMMsgObject)msgObj;
        String replyToQueueName = mqMsgObj.getReplyToQueueName();
        ...
    }
}
catch (Exception e)
{
    ...
}

```

MQeMQMMsgObject getReport**Syntax**

public int getReport() throws Exception

Description

This method returns the value of the Report header field.

Parameters

none

Return values

An int containing the Report value.

Exceptions**java.lang.Exception**

if there is an error reading the value from the message object

Example

```

try
{
    if (msgObj instanceof MQeMQMMsgObject)
    {
        MQeMQMMsgObject mqMsgObj = (MQeMQMMsgObject)msgObj;
        int report = mqMsgObj.getReport();
        ...
    }
}
catch (Exception e)
{
    ...
}

```

MQeMQMMsgObject getUserId**Syntax**

public String getUserId() throws Exception

Description

This method returns the value of the User Id header field.

Parameters

none

Return values

A String containing the value of the User Id.

MQeMQMsgObject

Exceptions

java.lang.Exception if there is an error reading the value from the message object

Example

```
try
{
    if (msgObj instanceof MQeMQMsgObject)
    {
        MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
        String userId = mqMsgObj.getUserId();
        ...
    }
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject setAccountingToken

Syntax

```
public void setAccountingToken(byte [] accountingToken) throws Exception
```

Description

This method sets the value of the Accounting Token header field in the MQSeries style message.

Parameters

accountingToken
a byte array containing the value to be set in the Accounting Token field

Return values

none

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    byte [] accountingToken = ...;
    mqeMsgObj.setAccountingToken(accountingToken);
    ...
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject setApplicationIdData

Syntax

```
public void setApplicationIdData(String applicationIdData) throws Exception
```

Description

This method sets the value of the Application Id Data header field in the MQSeries style message.

Parameters**applicationIdData**

a String containing the value to be set in the Application Id Data field.

Return values

none

Exceptions**java.lang.Exception**

if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMMsgObject mqeMsgObj = new MQeMQMMsgObject();
    String applIdData = ...;
    mqeMsgObj.setApplicationIdData(applIdData);
    ...
}
catch (Exception e)
{
    ...
}
```

MQeMQMMsgObject setApplicationOriginData**Syntax**

```
public void setApplicationOriginData(String applicationOriginData) throws Exception
```

Description

This method sets the value of the Application Origin Data header field in the MQSeries style message

Parameters**applicationOriginData**

a String containing the value to be set in the Application Origin Data field.

Return values

none

Exceptions**java.lang.Exception**

if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMMsgObject mqeMsgObj = new MQeMQMMsgObject();
    String applOriginData = ...;
    mqeMsgObj.setApplicationOriginData(applOriginData);
    ...
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject

MQeMQMsgObject setBackoutCount

Syntax

```
public void setBackoutCount(int backoutCount) throws Exception
```

Description

This method sets the value of the Backout Count header field in the MQSeries style message.

Parameters

backoutCount an int containing the value to be set in the Backout Count field

Return values

none

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    int backoutCount = ...;
    mqeMsgObj.setBackoutCount(backoutCount);
    ...
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject setCharacterSet

Syntax

```
public void setCharacterSet(int characterSet) throws Exception
```

Description

This method sets the value of the coded Character Set Identifier header field in the MQSeries style message

Parameters

characterSet an int containing the value to be set in the coded Character Set Identifier field

Return values

none

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    int characterSet = ...;
    mqeMsgObj.setCharacterSet(characterSet);
    ...
}
```

```

catch (Exception e)
{
    ...
}

```

MQeMQMMsgObject setCorrelationId

Syntax

```
public void setCorrelationId(byte [] correlationId) throws Exception
```

Description

This method sets the value of the Correlation Id header field in the MQSeries style message. It also sets the Correlation Id for use within the MQSeries Everyplace system itself

Parameters

correlationId a byte array containing the value to be set in the Correlation Id field.

Return values

none

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```

try
{
    MQeMQMMsgObject mqeMsgObj = new MQeMQMMsgObject();
    byte [] correlationId = ...;
    mqeMsgObj.setCorrelationId(correlationId);
    ...
}
catch (Exception e)
{
    ...
}

```

MQeMQMMsgObject setData

Syntax

```
public void setData(byte [] data) throws Exception
```

Description

This method sets the message data in the MQSeries style message.

Parameters

data if there is an error setting the value in the message object

Return values

none

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```

try
{
    MQeMQMMsgObject mqeMsgObj = new MQeMQMMsgObject();
    byte [] data = ...;

```

MQeMQMsgObject

```
        mqeMsgObj.setData(data);
        ...
    }
    catch (Exception e)
    {
        ...
    }
}
```

MQeMQMsgObject setEncoding

Syntax

```
public void setEncoding(int encoding) throws Exception
```

Description

This method sets the value of the Encoding header field in the MQSeries style message

Parameters

encoding an int containing the value to be set in the Encoding field

Return values

none

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    int encoding = ...;
    mqeMsgObj.setEncoding(encoding);
    ...
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject setExpiry

Syntax

```
public void setExpiry(int expiry) throws Exception
```

Description

This method sets the value of the Expiry header field in the MQSeries style message. It also sets the expiry time of the message for use within the MQSeries Everyplace system itself.

Parameters

expiry an int containing the value to be set in the expiry field. The value should be in tenths of a second, as is used in MQSeries messages (not in milliseconds, which is used for MQSeries Everyplace expiry time)..

Return values

none

Exceptions

java.lang.Exception

if there is an error setting the value in the message object

Example

```

try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    int expiry = ...;
    mqeMsgObj.setExpiry(expiry);
    ...}
catch (Exception e)
{
    ...
}

```

MQeMQMsgObject setFeedback**Syntax**

```
public void setFeedback(int feedback) throws Exception
```

Description

This method sets the value of the Feedback header field in the MQSeries style message.

Parameters

feedback an int containing the value to be set in the Feedback field.

Return values

none

Exceptions**java.lang.Exception**

if there is an error setting the value in the message object

Example

```

try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    int feedback = ...;
    mqeMsgObj.setFeedback(feedback);
    ...
}
catch (Exception e)
{
    ...
}

```

MQeMQMsgObject setFormat**Syntax**

```
public void setFormat(String format) throws Exception
```

Description

This method sets the value of the Format header field in the MQSeries style message

Parameters

format a String containing the value to be set in the Format field.

Return values

none

MQeMQMsgObject

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    String format = ...;
    mqeMsgObj.setFormat(format);
    ...
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject setGroupId

Syntax

```
public void setGroupId(byte [] groupId) throws Exception
```

Description

This method sets the value of the Group Id header field in the MQSeries style message.

Parameters

groupId a byte array containing the value to be set in the Group Id field.

Return values

none

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    byte [] groupId = ...;
    mqeMsgObj.setGroupId(groupId);
    ...
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject setMessageFlags

Syntax

```
public void setMessageFlags(int messageFlags) throws Exception
```

Description

This method sets the value of the Message Flags header field in the MQSeries style message

Parameters

format a String containing the value to be set in the Format field.

Return values

none

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    int messageFlags = ...;
    mqeMsgObj.setMessageFlags(messageFlags);
    ...
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject setMessageId**Syntax**

```
public void setMessageId(byte [] messageId) throws Exception
```

Description

This method sets the value of the Message Id header field in the MQSeries style message

Parameters

messageId a byte array containing the value to be set in the Message Id field.

Return values

none

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    byte [] messageId = ...;
    mqeMsgObj.setMessageId(messageId);
    ...
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject setMessageSequenceNumber**Syntax**

```
public void setMessageSequenceNumber(int seqNo) throws Exception
```

Description

This method sets the value of the Message Sequence Number header field in the MQSeries style message

MQeMQMsgObject

Parameters

seqNo an int containing the value to be set in the Message Sequence Number field.

Return values

none

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    int seqNo = ...;
    mqeMsgObj.setMessageSequenceNumber(seqNo);
    ...
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject setMessageType

Syntax

```
public void setMessageType(int messageType) throws Exception
```

Description

This method sets the value of the Message Type header field in the MQSeries style message. It also sets the message style for use within the MQSeries Everyplace system itself.

Parameters

messageType an int containing the value to be set in the Message Type field.

Return values

none

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    int messageType = ...;
    mqeMsgObj.setMessageType(messageType);
    ...
}
catch (Exception e)
{
    ...
}
```

MQeMQMMsgObject setOffset

Syntax

```
public void setOffset(int offset) throws Exception
```

Description

This method sets the value of the Offset header field in the MQSeries style message

Parameters

offset an int containing the value to be set in the Offset field

Return values

none

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMMsgObject mqeMsgObj = new MQeMQMMsgObject();
    int offset = ...;
    mqeMsgObj.setOffset(offset);
    ...
}
catch (Exception e)
{
    ...
}
```

MQeMQMMsgObject setOriginalLength

Syntax

```
public void setOriginalLength(int len) throws Exception
```

Description

This method sets the value of the Original Length header field in the MQSeries style message.

Parameters

len an int containing the value to be set in the Original Length field.

Return values

none

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMMsgObject mqeMsgObj = new MQeMQMMsgObject();
    int len = ...;
    mqeMsgObj.setOriginalLength(len);
    ...
}
```

MQeMQMsgObject

```
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject setPersistence

Syntax

```
public void setPersistence(int persistence) throws Exception
```

Description

This method sets the value of the Persistence header field in the MQSeries style message.

Parameters

persistence an int containing the value to be set in the Persistence field

Return values

none

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    int persistence = ...;
    mqeMsgObj.setPersistence(persistence);
    ...
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject setPriority

Syntax

```
public void setPriority(int priority) throws Exception
```

Description

This method sets the value of the Priority header field in the MQSeries style message. It also sets the priority of the message for use within the MQSeries Everyplace system itself.

Parameters

priority an int containing the value to be set in the Priority field. The value should be between 0 and 9 (inclusive).

Return values

none

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```

try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    int priority = ...;
    mqeMsgObj.setPriority(priority);
    ...
}
catch (Exception e)
{
    ...
}

```

MQeMQMsgObject setPutApplicationName

Syntax

```
public void setPutApplicationName(String putApplicationName) throws Exception
```

Description

This method sets the value of the Put Application Name header field in the MQSeries style message.

Parameters

putApplicationName

a String containing the value to be set in the Put Application Name field.

Return values

none

Exceptions

java.lang.Exception	if there is an error setting the value in the message object
----------------------------	--

Example

```

try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    String putAppName = ...;
    mqeMsgObj.setPutApplicationName(putAppName);
    ...
}
catch (Exception e)
{
    ...
}

```

MQeMQMsgObject setPutApplicationType

Syntax

```
public void setPutApplicationType(int putApplicationType) throws Exception
```

Description

This method sets the value of the Put Application Type header field in the MQSeries style message.

Parameters

putApplicationType

an int containing the value to be set in the Put Application Type field

Return values

none

MQeMQMsgObject

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    int putAppltype = ...;
    mqeMsgObj.setPutApplicationType(putApplType);
    ...
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject setPutDateTime

Syntax

```
public void setPutDateTime(GregorianCalendar calendar) throws Exception
```

Description

This method sets the value of the Put Date Time header field in the MQSeries style message. A GregorianCalendar object is used to specify the date and time, for consistency with the MQSeries Java Client.

Parameters

calendar a GregorianCalendar object containing the value to be set in the Put Date Time field.

Return values

none

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    GregorianCalendar calendar = ...;
    mqeMsgObj.setPutDateTime(calendar);
    ...
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject setReplyToQueueManagerName

Syntax

```
public void setReplyToQueueManagerName(String replyToQMName) throws Exception
```

Description

This method sets the value of the Reply To Queue Manager Name header field in the MQSeries style message.

Parameters

replyToQMName

a String containing the value to be set in the Reply To Queue Manager Name field.

Return values

none

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    String replyToQMName = ...;
    mqeMsgObj.setReplyToQueueManagerName(replyToQMName);
    ...
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject setReplyToQueueName**Syntax**

```
public void setReplyToQueueName(String replyToQueueName) throws Exception
```

Description

This method sets the value of the Reply To Queue Name header field in the MQSeries style message.

Parameters**replyToQueueName**

a String containing the value to be set in the Reply To Queue Name field.

Return values

none

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    String replyToQueueName = [];
    mqeMsgObj.setReplyToQueueName(replyToQueueName);
    ...
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject setReport**Syntax**

```
public void setReport(int report) throws Exception
```

MQeMQMsgObject

Description

This method sets the value of the Report header field in the MQSeries style message.

Parameters

report an int containing the value to be set in the Report field

Return values

none

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    int report = ...;
    mqeMsgObj.setReport(report);
    ...
}
catch (Exception e)
{
    ...
}
```

MQeMQMsgObject setUserId

Syntax

```
public void setUserId(String userId) throws Exception
```

Description

This method sets the value of the User Id header field in the MQSeries style message.

Parameters

userId a String containing the value to be set in the User Id field

Return values

none

Exceptions

java.lang.Exception if there is an error setting the value in the message object

Example

```
try
{
    MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
    String userId = ...;
    mqeMsgObj.setUserId(userId);
    ...
}
catch (Exception e)
{
    ...
}
```

Chapter 8. Classes in com.ibm.mqe.mqbridge

This section contains detailed information about the following MQSeries Everyplace classes and interfaces:

Table 18. Classes in package com.ibm.mqe.mqbridge

Class or Interface name	Purpose
MQeCharacteristicLabels	Groups together all the <i>labels</i> used in any MQeFields object used in the bridge code
MQeClientConnectionAdminMsg	Used to encapsulate an administration command that acts on the MQeClientConnection object.
MQeListenerAdminMsg	Used to encapsulate an administration command that acts on the MQeListener object.
MQeMQBridgeAdminMsg	Used to encapsulate an administration command that acts on the MQeBridge object.
MQeMQBridgeQueue	This queue is used as the interface to the MQS bridge
MQeMQBridgeQueueAdminMsg	Used to administer an MQBridge queue
MQeMQBridges	Loads and maintains all bridge objects associated with a given MQe server"
MQeMQBridgesAdminMsg	Used to encapsulate an administration command that acts on the MQeBridges object.
MQeMQMgrProxyAdminMsg	Used to encapsulate an administration command that acts on the MQeQMgrProxy object.
MQeRunState	Holds the <i>run state</i> of an administered object
MQeTransformerInterface	All classes that can transform MQMessages to MQeMsgObjects (and vice versa) must conform to this interface

Licence warning

The use of any of the classes in com.ibm.mqe.mqbridge requires a **gateway (server)** licence.

MQeCharacteristicLabels

This class groups together all the *labels* used in any **MQeFields** object used in the bridge code.

These labels are used by administration messages, and by the administered object runtime characteristics, and persistent registry entry objects.

Package **com.ibm.mqe.mqbridge**

This class extends **Object**

Constants and variables

MQeCharacteristicLabels provides the following constants and variables :

MQE_FIELD_LABEL_ADMINISTERED_OBJECT_CLASS

Syntax

```
public static final String MQE_FIELD_LABEL_ADMINISTERED_OBJECT_CLASS
```

Description

Used by all administered object registry entries. Indicates the class that should be loaded when you want to instantiate the administered object.

Example

```
String adminClass = fields.getUnicode(  
    MQeCharacteristicLabels.MQE_FIELD_LABEL_ADMINISTERED_OBJECT_CLASS);
```

MQE_FIELD_LABEL_AFFECT_CHILDREN

Syntax

```
public static final String MQE_FIELD_LABEL_AFFECT_CHILDREN
```

Description

Used to control whether a start or delete action affects child objects when applied to an administered object.

Example

```
myFields.putBoolean(  
    MQeCharacteristicLabels.MQE_FIELD_LABEL_AFFECT_CHILDREN, true );
```

MQE_FIELD_LABEL_BRIDGE_NAME

Syntax

```
public static final String MQE_FIELD_LABEL_BRIDGE_NAME
```

Description

Example

```
myFields.putUnicode(MQeCharacteristicLabels.MQE_FIELD_LABEL_BRIDGE_NAME,  
    "MQBridgeV100");
```

MQE_FIELD_LABEL_CCSID

Syntax

```
public static final String MQE_FIELD_LABEL_CCSID
```

Description

The label for the field holding the CCSID used on the underlying MQSeries Java client channel.

Example

```
fields.putInt( MQeCharacteristicLabels.MQE_FIELD_LABEL_CCSD , 819 );
```

MQE_FIELD_LABEL_CHILD**Syntax**

```
public static final String MQE_FIELD_LABEL_CHILD
```

Description

The label of the MQSeries Everyplace field that holds the name of the child of an administered object

Example

```
String childName = myFields.getUnicode(
    MQeCharacteristicLabels.MQE_FIELD_LABEL_CHILD );
```

MQE_FIELD_LABEL_CHILDREN**Syntax**

```
public static final String MQE_FIELD_LABEL_CHILDREN
```

Description

The label associated with the children of the MQSeries Everyplace field. The value of the field holds a list of proxy object names

Example

```
MQeFields[] children = bridgesDetails.getFieldsArray(
    MQeCharacteristicLabels.MQE_FIELD_LABEL_CHILDREN );
```

MQE_FIELD_LABEL_CLIENT_CONNECTION_NAME**Syntax**

```
public static final String MQE_FIELD_LABEL_CLIENT_CONNECTION_NAME
```

Description

Label for the MQSeries Everyplace field that holds the name of the client connection that the administration message is being sent to.

Example

```
fields.putUnicode(
    MQeCharacteristicLabels.MQE_FIELD_LABEL_CLIENT_CONNECTION_NAME,
    "SYSTEM.DEF.SVRCONN");
```

MQE_FIELD_LABEL_DEAD_LETTER_Q_NAME**Syntax**

```
public static final String MQE_FIELD_LABEL_DEAD_LETTER_Q_NAME
```

Description

Label for the field holding the name of the dead letter queue on the MQSeries system.

Example

```
fields.putUnicode(
    MQeCharacteristicLabels.MQE_FIELD_LABEL_DEAD_LETTER_Q_NAME,
    "MQE.DLQ" );
```

MQeCharacteristicLabels

MQE_FIELD_LABEL_DEFAULT_TRANSFORMER

Syntax

```
public static final String MQE_FIELD_LABEL_DEFAULT_TRANSFORMER
```

Description

Label holding the name of the MQSeries Everyplace field that holds the name of the default transformer class that is used to convert from MQSeries Everyplace to MQSeries and from MQSeries to MQSeries Everyplace message formats. This value is used if no transformer is specified on the target queue definition.

Example

```
fields.putUnicode(  
    MQeCharacteristicLabels.MQE_FIELD_LABEL_DEFAULT_TRANSFORMER,  
    "com.ibm.mqe.mqbridge.MQeBaseTransformer");
```

MQE_FIELD_LABEL_FLOWS_PER_COMMIT

Syntax

```
public static final String MQE_FIELD_LABEL_FLOWS_PER_COMMIT
```

Description

Label indicating how many times through the listener flows the listener is allowed to go before the sync queue on the MQSeries system is cleaned up. This is only relevant when the listener is using the MQSeries sync queue as it's state store.

Example

```
fields.putInt(  
    MQeCharacteristicLabels.MQE_FIELD_LABEL_FLOWS_PER_COMMIT , 100 );
```

MQE_FIELD_LABEL_HEARTBEAT_INTERVAL

Syntax

```
public static final String MQE_FIELD_LABEL_HEARTBEAT_INTERVAL
```

Description

The label holding the value of the time interval (in units of 1minute) that dictates the period of each "heartbeat" event coming from the **MQeHeart** class. This affects the accuracy of any timer mechanisms in the bridge, as these all use the heartbeat as a "tick" of their timer.

Example

```
fields.putInt(  
    MQeCharacteristicLabels.MQE_FIELD_LABEL_HEARTBEAT_INTERVAL, 5);
```

MQE_FIELD_LABEL_HOST_NAME

Syntax

```
public static final String MQE_FIELD_LABEL_HOST_NAME
```

Description

MQSeries Everyplace field label for the hostname field.

Example

```
// Using an IP address  
fields.putUnicode(MQeCharacteristicLabels.MQE_FIELD_LABEL_HOST_NAME,  
    "127.0.0.1");  
// Using a host in the default domain
```

MQeCharacteristicLabels

```
fields.putUnicode(MQeCharacteristicLabels.MQE_FIELD_LABEL_HOST_NAME,
    "lizzie");
// Using a fully qualified hostname
fields.putUnicode(MQeCharacteristicLabels.MQE_FIELD_LABEL_HOST_NAME,
    "daytona2.hursley.ibm.com");
// Using the MQSeries Java Bindings on localhost
fields.putUnicode(MQeCharacteristicLabels.MQE_FIELD_LABEL_HOST_NAME,
    "");
```

MQE_FIELD_LABEL_LISTENER_NAME

Syntax

```
public static final String MQE_FIELD_LABEL_LISTENER_NAME
```

Description

Label for the field holding the MQSeries transmission queue listener name (the name of the transmit queue on MQSeries).

Example

```
fields.putUnicode( MQeCharacteristicLabels.MQE_FIELD_LABEL_LISTENER_NAME,
    "MQE.XMITQ" );
```

MQE_FIELD_LABEL_LISTENER_STATE_STORE_ADAPTER

Syntax

```
public static final String MQE_FIELD_LABEL_LISTENER_STATE_STORE_ADAPTER
```

Description

Label holding the name of the MQSeries Everyplace field that holds the name of the adapter class used by the listener to store its state.

Example

```
fields.putUnicode(
    MQeCharacteristicLabels.MQE_FIELD_LABEL_LISTENER_STATE_STORE_ADAPTER,
    "com.ibm.mqe.adapters.MQeDiskFieldsAdapter" );
```

MQE_FIELD_LABEL_MAX_CONNECTION_IDLE_TIME

Syntax

```
public static final String MQE_FIELD_LABEL_MAX_CONNECTION_IDLE_TIME
```

Description

The label for the field holding the maximum time value that an MQSeries client connection is kept open if it is idle. Connections that are idle for more than this time are closed by the bridge.

Example

```
fields.putInt(
    MQeCharacteristicLabels.MQE_FIELD_LABEL_MAX_CONNECTION_IDLE_TIME , 5 );
```

MQE_FIELD_LABEL_MQ_BRIDGE_ADAPTER_CLASS

Syntax

```
public static final String MQE_FIELD_LABEL_MQ_BRIDGE_ADAPTER_CLASS
```

Description

The label for the field holding the name of the bridge adapter class. The bridge adapter is a Java class that allows messages sent to an MQSeries bridge queue to be moved to the MQSeries system.

MQeCharacteristicLabels

Example

```
fields.putUnicode(  
    MQeCharacteristicLabels.MQE_FIELD_LABEL_MQ_BRIDGE_ADAPTER_CLASS,  
    "com.ibm.mqe.mqbridge.MQeMQAdapter" );
```

MQE_FIELD_LABEL_MQ_Q_MGR_PROXY_NAME

Syntax

```
public static final String MQE_FIELD_LABEL_MQ_Q_MGR_PROXY_NAME
```

Description

Label of the MQSeries Everyplace field holding the name of the MQSeries queue manager proxy object.

Example

```
String name = myFields.getUnicode(  
    MQeCharacteristicLabels.MQE_FIELD_LABEL_MQ_Q_MGR_PROXY_NAME);
```

MQE_FIELD_LABEL_NAME

Syntax

```
public static final String MQE_FIELD_LABEL_NAME
```

Description

Label holding the name of the MQSeries Everyplace field that holds the name of the bridge, proxy, client connection, or listener.

Example

```
String fieldName = myFields.getUnicode(  
    MQeCharacteristicLabels.MQE_FIELD_LABEL_NAME );
```

MQE_FIELD_LABEL_PASSWORD

Syntax

```
public static final String MQE_FIELD_LABEL_PASSWORD
```

Description

Label holding the password used with the userid when the bridge talks to MQSeries .

Example

```
fields.putUnicode(  
    MQeCharacteristicLabels.MQE_FIELD_LABEL_PASSWORD, "chocolate" );
```

MQE_FIELD_LABEL_PORT

Syntax

```
public static final String MQE_FIELD_LABEL_PORT
```

Description

Label for the MQSeries Everyplace field that holds the port number of the MQSeries channel listener

Example

```
fields.putInt( MQeCharacteristicLabels.MQE_FIELD_LABEL_PORT , 1414 );
```

MQE_FIELD_LABEL_RECEIVE_EXIT

Syntax

```
Public static final String MQE_FIELD_LABEL_RECEIVE_EXIT
```

Description

Label for the field holding the receive exit used on the underlying MQSeries Java client channel.

Example

```
fields.putUnicode( MQeCharacteristicLabels.MQE_FIELD_LABEL_RECEIVE_EXIT,
    "my.ReceiveExit" );
```

MQE_FIELD_LABEL_RUN_STATE**Syntax**

```
public static final String MQE_FIELD_LABEL_RUN_STATE
```

Description

The label of the MQSeries Everyplace field that holds the snapshot of the current state of the administered object.

Example

```
fields.putInt( MQeCharacteristicLabels.MQE_FIELD_LABEL_RUN_STATE,
    MQeRunState.RUN_STATE_RUNNING );
```

MQE_FIELD_LABEL_SECONDS_WAIT_FOR_MSG**Syntax**

```
public static final String MQE_FIELD_LABEL_SECONDS_WAIT_FOR_MSG
```

Description

Label for the field holding the seconds that the MQSeries transmission queue listener uses in the MQSeries Java client GetMessage(wait) method. (For development use only).

Example

```
fields.putInt(
    MQeCharacteristicLabels.MQE_FIELD_LABEL_SECONDS_WAIT_FOR_MSG, 10 );
```

MQE_FIELD_LABEL_SECURITY_EXIT**Syntax**

```
public static final String MQE_FIELD_LABEL_SECURITY_EXIT
```

Description

Label for the field holding the security exit used on the underlying MQSeries Java client channel.

Example

```
fields.putUnicode( MQeCharacteristicLabels.MQE_FIELD_LABEL_SECURITY_EXIT,
    "my.SecurityExit" );
```

MQE_FIELD_LABEL_SEND_EXIT**Syntax**

```
public static final String MQE_FIELD_LABEL_SEND_EXIT
```

Description

Label for the field holding the send exit used on the underlying MQSeries Java client channel.

Example

```
fields.putUnicode( MQeCharacteristicLabels.MQE_FIELD_LABEL_SEND_EXIT,
    "my.SendExit" );
```

MQeCharacteristicLabels

MQE_FIELD_LABEL_STARTUP_RULE_CLASS

Syntax

```
public static final String MQE_FIELD_LABEL_STARTUP_RULE_CLASS
```

Description

Used by all the administered object registry entries. Indicates the class that should be used to decide whether or not the administered object is started up when it is loaded

Example

```
String ruleClass = fields.getUnicode(  
    MQeCharacteristicLabels.MQE_FIELD_LABEL_STARTUP_RULE_CLASS);
```

MQE_FIELD_LABEL_SYNC_Q_NAME

Syntax

```
public static final String MQE_FIELD_LABEL_SYNC_Q_NAME
```

Description

MQSeries Everyplace field label for the SyncQName field. Used to keep track of the state of progress through the assured delivery flows.

Example

```
fields.putUnicode( MQeCharacteristicLabels.MQE_FIELD_LABEL_SYNC_Q_NAME,  
    "SYNC.Q" );
```

MQE_FIELD_LABEL_SYNC_Q_PURGE_INTERVAL

Syntax

```
public static final String MQE_FIELD_LABEL_SYNC_Q_PURGE_INTERVAL
```

Description

The time interval between successive purges of the sync queue

Example

```
fields.putInt(  
    MQeCharacteristicLabels.MQE_FIELD_LABEL_SYNC_Q_PURGE_INTERVAL , 60 );
```

MQE_FIELD_LABEL_SYNC_Q_PURGER_RULES_CLASS

Syntax

```
public static final String MQE_FIELD_LABEL_SYNC_Q_PURGER_RULES_CLASS
```

Description

MQSeries Everyplace field label for the SyncQPurgerRulesClass field

Example

```
fields.putUnicode(  
    MQeCharacteristicLabels.MQE_FIELD_LABEL_SYNC_Q_PURGER_RULES_CLASS,  
    "com.ibm.mqe.mqbridge.MQeSyncQueuePurgerRule" );
```

MQE_FIELD_LABEL_TRANSFORMER

Syntax

```
public static final String MQE_FIELD_LABEL_TRANSFORMER
```


MQeCharacteristicLabels

Description

Label indicating which transformer class should be used to convert the MQSeries message to an MQSeries Everyplace message before it is dispatched to the MQSeries Everyplace network.

Example

```
fields.putUnicode( MQeCharacteristicLabels.MQE_FIELD_LABEL_TRANSFORMER,  
                  "com.ibm.mqe.mqbridge.MQeBaseTransformer" );
```

MQE_FIELD_LABEL_UNDELIVERED_MESSAGE_RULE_CLASS

Syntax

```
public static final String MQE_FIELD_LABEL_UNDELIVERED_MESSAGE_RULE_CLASS
```

Description

Label for the field holding the Java class name of the rule to use when the message cannot be delivered to its destination.

Example

```
fields.putUnicode(  
    MQeCharacteristicLabels.MQE_FIELD_LABEL_UNDELIVERED_MESSAGE_RULE_CLASS,  
    "com.ibm.mqe.mqbridge.MQeUndeliveredMessageRule" );
```

MQE_FIELD_LABEL_USER_ID

Syntax

```
public static final String MQE_FIELD_LABEL_USER_ID
```

Description

Label for the userid assumed by the bridge when talking to MQSeries

Example

```
fields.putUnicode( MQeCharacteristicLabels.MQE_FIELD_LABEL_USER_ID, "mqm" );
```

MQeCharacteristicLabels

Syntax

```
public MQeCharacteristicLabels()
```

Description

Creates an MQeCharacteristicLabels object

Parameters

none

Return values

none

Exceptions

none

Example

MQeClientConnectionAdminMsg

This is a special type of MQSeries Everyplace message that is used to encapsulate an administration command. The message is created by the application that is doing the administration.

The logic performed on the target MQSeries Everyplace system is also in this class.

The administration queue invokes the performAction method.

Package **com.ibm.mqe.mqbridge**

This class extends **MQeMQQMgrProxyAdminMsg**

Constants and variables

MQeQueueAdminMsg provides the following constants and variables in addition to those provided by **MQeMQQMgrProxyAdminMsg**:

Constructor summary

Constructor	Purpose
MQeClientConnectionAdminMsg	Creates and initializes an MQeClientConnectionAdminMsg.

Method summary

Method	Purpose
characteristics	Creates an MQSeries Everyplace fields object containing all the MQe fields required for an administration message of this type.
getClientConnectionName	Gets the client connection name from the administered object.
getName	Gets the name of the object to be administered.
putClientConnectionName	Puts the client connection name in an MQSeries Everyplace field in the MQSeries Everyplace fields administration message object.
setName	Puts the name information in an MQSeries Everyplace field in the MQe fields administration message object and also sets the name of the MQSeries Everyplace queue manager that is associated with this bridge.

MQeClientConnectionAdminMsg

Syntax

1. `public MQeClientConnectionAdminMsg() throws Exception`
2. `public MQeClientConnectionAdminMsg(String bridgeName, String nameOfMQMgrProxy, String clientConnectionName, boolean affectChildren) throws Exception`

Description

There are two constructors:

MQeClientConnectionAdminMsg

1. This version creates and initializes a default MQeClientConnectionAdminMsg
2. This version includes fields that are needed to initialize the administration message. It does not include the action that the administration message will hold

Parameters

- bridgeName** A String containing the name of the bridge to which the administration message is directed. If set to *null*, or "", it is not set.
- nameOfMQMgrProxy** A String containing the name of the proxy to which the administration message is directed. If set to *null*, or "", it is not set.
- clientConnectionName** A String containing the name of the client connection to which the administration message is directed. If set to *null*, or "", it is not set.
- affectChildren** A boolean flag indicating whether or not this administration message affects all the children. This is only applicable if the action is **start** or **delete**.

Return Values

none

Exceptions

Fails if any of the parameters contain invalid characters

Example

```
MQeClientConnectionAdminMsg msg ;
msg = new MQeClientConnectionAdminMsg( "ExampleQM.MQBridgeV100"
    , "MQA"
    , "MQ.to.ExampleQM"
    , false
    );
```

MQeClientConnectionAdminMsg characteristics

Syntax

```
public MQeFields characteristics() throws Exception
```

Description

Creates an **MQeFields** object containing all the fields required for an administration message of this type.

Overrides **characteristics** in class **MQeMQMgrProxyAdminMsg** .

Parameters

none

Return values

An **MQeFields** object containing the characteristics of the resource. The complete set of field names and types for the resource can be determined from the resulting fields object.

Exceptions

java.lang.Exception If the **MQeFields** object cannot be created

Example

MQeClientConnectionAdminMsg

```
MQeClientConnectionAdminMsg msg;  
msg = new MQeClientConnectionAdminMsg( "ExampleQM.MQBridgeV100",  
                                       "MQA",  
                                       "MQ.to.ExampleQM",  
                                       false);  
MQeFields cconAdminCharacteristics = msg.characteristics();
```

MQeClientConnectionAdminMsg getClientConnectionName

Syntax

```
public String getClientConnectionName() throws Exception
```

Description

Gets the client connection name from the administered object. This method can be issued against an MQeClientConnectionAdminMsg or one of its descendants.

Parameters

none

Return values

The name of the client connection to which this administration message is to be sent.

Exceptions

java.lang.Exception	If the name has not been set in this administration message, or if the name that has been set is invalid
----------------------------	--

Example

```
MQeClientConnectionAdminMsg msg;  
msg = new MQeClientConnectionAdminMsg( "ExampleQM.MQBridgeV100",  
                                       "MQA",  
                                       "MQ.to.ExampleQM",  
                                       false);  
String cconName = msg.getClientConnectionName();
```

MQeClientConnectionAdminMsg getName

Syntax

```
public String getName()
```

Description

Gets the name of the client connection that is to be administered. In this case it's the name of the client connection that has been set by the setName or putClientConnectionName methods. When issued against an object of this class it is identical to getClientConnectionName().

Overrides **getName** in class **MQeMQQMGrProxyAdminMsg**.

Parameters

none

Return values

A String containing the name of the administered object we want to create, or *null* if the name is not set.

Exceptions

none

Example

MQeClientConnectionAdminMsg

```
MQeClientConnectionAdminMsg msg;  
msg = new MQeClientConnectionAdminMsg( "ExampleQM.MQBridgeV100",  
                                       "MQA",  
                                       "MQ.to.ExampleQM",  
                                       false);  
  
String cconName = msg.getName();
```

MQeClientConnectionAdminMsg putClientConnectionName

Syntax

```
public void putClientConnectionName(String clientConnectionName)  
                                   throws Exception
```

Description

This is used to add the MQSeries queue manager name to the administration message. It puts the client connection name in an MQSeries Everyplace field in the MQSeries Everyplace fields administration message object.

Parameters

clientConnectionName

A String containing the name of the client connection to which the administration message is directed. This string is validated using the `validateName()` method to make sure it contains only legal characters.

Return values

none

Exceptions

java.lang.Exception

If there are any invalid characters in the name parameters

Example

```
MQeClientConnectionAdminMsg msg = new MQeClientConnectionAdminMsg();  
msg.setName( "ExampleQM.MQBridgeV100" ,  
            "MQA" ,  
            "MQ.to.ExampleQM" );
```

MQeClientConnectionAdminMsg setName

Syntax

```
public void setName(String bridgeName,  
                   String mqQMgrProxyName,  
                   String clientConnectionName) throws Exception
```

Description

Puts the name information in an MQSeries Everyplace field in the MQSeries Everyplace fields administration message object and also sets the name of the MQSeries Everyplace queue manager that is associated with this bridge.

Parameters

bridgeName A String containing the name of the bridge to which the administration message is directed. If set to *null*, or "", it is not set.

MQeClientConnectionAdminMsg

mqQMgrProxyName

A String containing the name of the proxy to which the administration message is directed. If set to *null*, or "", it is not set.

clientConnectionName

A String containing the name of the client connection to which the administration message is directed. If set to *null*, or "", it is not set.

Return values

none

Exceptions

java.lang.Exception

If there are any invalid characters in the name parameters.

Example

```
MQeClientConnectionAdminMsg msg = new MQeClientConnectionAdminMsg();
msg.setName( "ExampleQM.MQBridgeV100" ,
            "MQA" ,
            "MQ.to.ExampleQM" )
```

MQeListenerAdminMsg

This is a special type of MQSeries Everyplace message used to encapsulate an administration command. The message is created by the application that is doing the administration.

The logic performed on the target MQSeries Everyplace system is also in this class.

The administration queue invokes the `performAction` method.

Package **com.ibm.mqe.mqbridge**

This class extends **MQeClientConnectionAdminMsg**

Constructor summary

Constructor	Purpose
MQeListenerAdminMsg	Creates and initializes an MQeListenerAdminMsg.

Method summary

Method	Purpose
characteristics	Creates an MQSeries Everyplace fields object containing all the MQe fields required for an administration message of this type.
getListenerName	Gets the listener name from the administered object.
getName	Gets the name of the administered object to be created.
putListenerName	Puts the listener name in an MQSeries Everyplace field in the MQSeries Everyplace fields administration message object.
setName	Puts the name information in an MQSeries Everyplace field in the MQe fields administration message object and also sets the name of the MQSeries Everyplace queue manager that is associated with this bridge.

MQeListenerAdminMsg

Syntax

1. `public MQeListenerAdminMsg() throws Exception`
2. `public MQeListenerAdminMsg(String bridge, String MQMGrProxy, String clientConnection, String listener, boolean affectChildren) throws Exception`

Description

There are two constructors.

1. This version creates and initializes a default MQeListenerAdminMsg
2. This version includes the MQSeries Everyplace queue manager name, the name of the bridge, the name of the proxy, the name of the client connection, and the name of the listener

Parameters

bridge A String containing the name of the bridge to which the administration message is directed. If set to *null*, or "", it is not set.

MQeListenerAdminMsg

MQQMgrProxy

A String containing the name of the MQS queue manager that owns the transmission queue that the listener is set up to read from. If set to *null*, or "", it is not set.

clientConnection

A String containing the name of the client connection used to talk to the MQS queue manager. If set to *null*, or "", it is not set.

listener

A string containing the name of the listener. This matches the name of the transmission queue on MQS to which the listener "listens", for messages to be ready to move to the MQSeries Everyplace network.

affectChildren

A boolean flag indicating whether or not this administration message affects the children of the listener.

Return Values

none

Exceptions

Fails if any of the parameters contain invalid characters

Example

```
MQeListenerAdminMsg msg = new MQeListenerAdminMsg( "MQBridgeV100",
                                                    "lizzieQM",
                                                    "svrconn",
                                                    "MQE.XMITQ",
                                                    true);
```

MQeListenerAdminMsg characteristics

Syntax

```
public MQeFields characteristics() throws Exception
```

Description

Creates an MQSeries Everyplace fields object containing all the MQSeries Everyplace fields required for an administration message of this type.

Returns a fields object containing the characteristics of the resource. The complete set of field names and types for the resource can be determined from the resulting fields object.

Overrides **characteristics** in class **MQeClientConnectionAdminMsg**.

Parameters

none

Return values

An **MQeFields** object containing the characteristics of the resource.

Exceptions

java.lang.Exception

If the **MQeFields** object cannot be created

Example

```
MQeListenerAdminMsg msg = new MQeListenerAdminMsg( "MQBridgeV100",
                                                    "lizzieQM",
                                                    "svrconn",
                                                    "MQE.XMITQ",
                                                    true );

MQeFields characteristics = msg.characteristics();
```


MQeListenerAdminMsg getListenerName

Syntax

```
public String getListenerName() throws Exception
```

Description

Gets the listener name from the administered object.

Can be issued only against an MQeListenerAdminMsg object.

Parameters

none

Return values

The name of the listener.

Exceptions

java.lang.Exception

If the name has not been set in this administration message, or if the name that has been set is invalid

Example

```
MQeListenerAdminMsg msg = new MQeListenerAdminMsg( "MQBridgeV100",
                                                    "lizzieQM",
                                                    "svrconn",
                                                    "MQE.XMITQ",
                                                    true);

String listenerName = msg.getListenerName();
```

MQeListenerAdminMsg getName

Syntax

```
public String getName()
```

Description

Gets the name of the client connection that is to be administered.

When issued against an object of this class it is identical to `getListenerName()`.

Overrides `getName` in class `MQeClientConnectionAdminMsg`.

Parameters

none

Return values

A String containing the name of the administered object to be created, or *null* if the name is not set.

Exceptions

none

Example

```
MQeListenerAdminMsg msg = new MQeListenerAdminMsg( "MQBridgeV100",
                                                    "lizzieQM",
                                                    "svrconn",
                                                    "MQE.XMITQ",
                                                    true);

String listenerName = msg.getName();
```

MQeListenerAdminMsg

MQeListenerAdminMsg putListenerName

Syntax

```
public void putListenerName(String listener) throws Exception
```

Description

Used to add the MQS Queue Manager name to the administration message. Puts the listener name in an MQSeries Everyplace field in the MQSeries Everyplace fields administration message object.

This method is used by the source of the administration message.

Parameters

listener A string containing the name of the listener. This matches the name of the transmission queue on MQS to which the listener "listens", for messages to be ready to move to the MQSeries Everyplace network.

Return values

none

Exceptions

java.lang.Exception If there are any invalid characters in the name parameters.

Example

```
MQeListenerAdminMsg msg = new MQeListenerAdminMsg();  
msg.putListenerName("MQE.XMITQ");
```

MQeListenerAdminMsg setName

Syntax

```
public void setName(String bridge,  
                   String mqMgrProxy,  
                   String clientConnection  
                   String listener) throws Exception
```

Description

Puts the name information in an MQSeries Everyplace field in the MQSeries Everyplace fields administration message object and also sets the name of the MQSeries Everyplace queue manager that is associated with this bridge.

This method is used by the source of the administration message.

Parameters

bridge A String containing the name of the bridge to which the administration message is directed. If set to *null*, or "", it is not set.

MQMgrProxy

A String containing the name of the MQS queue manager that owns the transmission queue the listener is set up to read from. If set to *null*, or "", it is not set.

clientConnection

A String containing the name of the client connection used to talk to the MQSeries queue manager. If set to *null*, or "", it is not set.

listener A string containing the name of the listener. This matches

MQeListenerAdminMsg

the name of the transmission queue on MQSeries to which the listener "listens", for messages to be ready to move to the MQSeries Everyplace network.

Return values

none

Exceptions

java.lang.Exception

If there are any invalid characters in the name parameters.

Example

```
MQeListenerAdminMsg msg = new MQeListenerAdminMsg();  
msg.setName("MQBridgeV100", "lizzieQM", "svrconn", "MQE.XMITQ");
```

MQeMQBridgeAdminMsg

This is a special type of MQSeries Everyplace message that is used to encapsulate an administration command. The message is created by the application that is doing the administration.

The logic performed on the target MQSeries Everyplace system is also in this class.

The administration queue invokes the performAction method.

Package **com.ibm.mqe.mqbridge**

This class extends **MQeMQBridgesAdminMsg**

Constants and variables

MQeQueueAdminMsg provides the following constants and variables in addition to those provided by **MQeMQBridgesAdminMsg**:

DEFAULT_MQBRIDGE_NAME
public static final String DEFAULT_MQBRIDGE_NAME

Constructor summary

Constructor	Purpose
MQeMQBridgeAdminMsg	Creates and initializes an MQeBridgeAdminMsg.

Method summary

Method	Purpose
characteristics	Creates an MQSeries Everyplace fields object containing all the MQSeries Everyplace fields required for an administration message of this type.
create	Causes this administration message to be a "create" message.
delete	Causes this administration message to be a "delete" message.
getBridgeName	Gets the bridge name from the administered object.
getName	Gets the name of the administered object to be created.
putBridgeName	Puts the bridge connection name in an MQSeries Everyplace field in the MQSeries Everyplace fields administration message object.
setName	Puts the name information in an MQSeries Everyplace field in the MQSeries Everyplace fields administration message object and also sets the name of the MQSeries Everyplace queue manager that is associated with this bridge.

MQeMQBridgeAdminMsg

Syntax

1. public MQeMQBridgeAdminMsg() throws Exception
2. public MQeMQBridgeAdminMsg(String bridgeName, boolean affectChildren) throws Exception

Description

There are two constructors.

1. This version creates and initializes a default MQeBridgeAdminMsg
2. This version includes the name of the bridge and a flag to determine whether children should be affected by the administration commands

Parameters

- bridge** A String containing the name of the bridge to which the administration message is directed. If set to *null*, or "", it is not set.
- affectChildren** A boolean flag indicating whether or not this administration message affects the children of the listener.

Return Values

none

Exceptions

- java.lang.Exception** If any of the parameters contain invalid characters

Example

1. MQeMQBridgeAdminMsg msg = new MQeMQBridgeAdminMsg();
2. public MQeMQBridgeAdminMsg(java.lang.String bridge, boolean affectChildren) Exception

MQeMQBridgeAdminMsg characteristics**Syntax**

```
public MQeFields characteristics() throws Exception
```

Description

Creates an MQSeries Everyplace fields object containing all the MQSeries Everyplace fields required for an administration message of this type.

Returns a fields object containing the characteristics of the resource. The complete set of field names and types for the resource can be determined from the resulting fields object.

Overrides **characteristics** in class **MQeBridgesAdminMsg** .

Parameters

none

Return values

An MQeFields object containing the characteristics of the resource.

Exceptions

- java.lang.Exception** If the **MQeFields** object cannot be created.

Example

```
MQeMQBridgeAdminMsg msg = new MQeMQBridgeAdminMsg("MQBridgeV100", true);
MQeFields bridgeCharacteristics = msg.characteristics();
```

MQeMQBridgeAdminMsg create**Syntax**

```
public void create(MQeFields parms) throws Exception
```

Description

MQeMQBridgeAdminMsg

Used by the source of the administration message.

Causes this administration message to be a "create" message. When the target MQSeries Everyplace system processes this message, it creates an MQSeries queue manager entry.

Overrides **create** in class **MQeAdminMsg**.

Parameters

parms Any extra parameters you want to add to the message, or *null*.

Return values

none

Exceptions

java.lang.Exception If the **MQeFields** object cannot be created

Example

```
// Form a message that will create a new MQBridge.  
MQeMQBridgeAdminMsg msg = new MQeMQBridgeAdminMsg();  
msg.create( new MQeFields() );
```

MQeMQBridgeAdminMsg getBridgeName

Syntax

```
public String getBridgeName() throws Exception
```

Description

Gets the bridge name from the administered object.

Can be issued against an MQeMQBridgeAdminMsg or one of its descendants.

Parameters

none

Return values

The name of the bridge.

Exceptions

java.lang.Exception If the name has not been set in this administration message, or if the name that has been set is invalid

Example

```
MQeMQBridgeAdminMsg msg = new MQeMQBridgeAdminMsg("MQBridgeV100", true);  
String bridgeName = msg.getBridgeName();
```

MQeMQBridgeAdminMsg getName

Syntax

```
public String getName()
```

Description

Gets the name of the bridge that is to be administered. When issued against an object of this class it is identical to `getBridgeName()`.

Overrides `getName` in class `MQeBridgesAdminMsg`.

Parameters

none

Return values

A String containing the name of the administered object to be created, or *null* if the name is not set.

Exceptions

none

Example

```
MQeMQBridgeAdminMsg msg = new MQeMQBridgeAdminMsg("MQBridgeV100", true);
String bridgeName = msg.getName();
```

MQeMQBridgeAdminMsg delete

Syntax

1. `public void delete(MQeFields parms, boolean affectChildren) throws Exception`
2. `public void delete(MQeFields parms) throws Exception`

Description

There are two versions of this method:

1. This version is used by the source of the administration message and it causes this administration message to be a "delete" message. When the target MQSeries Everyplace system processes this message, it finds the specified bridge object, and deletes it. This operation is inherited by the other bridge object types.
2. This version is equivalent to `delete((MQeFields) parms, false)`. When the target MQSeries Everyplace system processes this message, it finds the specified bridge object, and deletes it. This operation is inherited by the other bridge object types.

Overrides `delete` in class `MQeAdminMsg`

Parameters

- parms** Any extra parameters you want to add to the message, or *null*.
- affectChildren** A boolean flag indicating whether or not this administration message affects the children of the listener.

Return values

none

Exceptions

- java.lang.Exception** If the delete fails

Example

1. // Form a message that will delete an MQBridge and its children.

```
MQeMQBridgeAdminMsg msg = new MQeMQBridgeAdminMsg();
msg.delete( new MQeFields(), true );
```
2. // Form a message that will delete an MQBridge.

```
MQeMQBridgeAdminMsg msg = new MQeMQBridgeAdminMsg();
msg.delete( new MQeFields() );
```

MQeMQBridgeAdminMsg

MQeMQBridgeAdminMsg putBridgeName

Syntax

```
public void putBridgeName(String bridge) throws Exception
```

Description

Used by the source of the administration message to add the MQSeries queue manager name to the administration message.

Puts the bridge name in an MQSeries Everyplace field in the MQSeries Everyplace fields administration message object.

Parameters

bridge A String containing the name of the bridge to which the administration message is directed. If set to *null*, or "", it is not set.

Return values

none

Exceptions

java.lang.Exception If there are any invalid characters in the name parameters

Example

```
MQeMQBridgeAdminMsg msg = new MQeMQBridgeAdminMsg();  
msg.putBridgeName("MQBridgeV100");
```

MQeMQBridgeAdminMsg setName

Syntax

```
public void setName(String bridge) throws Exception
```

Description

Used by the source of the administration message to add the bridge name to the administration message.

Puts the name information in an MQSeries Everyplace field in the MQSeries Everyplace fields administration message object and also sets the name of the MQSeries Everyplace queue manager that is associated with this bridge.

Overrides **setName** in class **MQAdminMsg**

Parameters

bridge A String containing the name of the bridge to which the administration message is directed. If set to *null*, or "", it is not set.

Return values

none

Exceptions

java.lang.Exception If there are any invalid characters in the name parameters

Example

```
MQeMQBridgeAdminMsg msg = new MQeMQBridgeAdminMsg();  
msg.setName("MQBridgeV100");
```


MQeMQBridgeQueue

This queue is used as the interface to the MQSeries bridge and is passed into the user-written "transformer" code.

The transformer is responsible for conversion from MQSeries Everyplace to MQSeries message formats, and vice-versa, but a reference to the MQeMQBridgeQueue class is passed to the transformer code only when the message is moving from MQSeries Everyplace to MQSeries.

The class holds details that the user-written transformer class implementation may wish to use when performing a transform operation on the message data.

Package **com.ibm.mqe.mqbridge**

This class extends **MQeRemoteQueue**

Method summary

Method	Purpose
getMQQMgr	Gets the name of the MQS queue manager that is used to convey messages onto the MQS network.
getQueueAttribute	Gets the attribute attached to the bridge queue.
getQueueManagerName	Gets the owning queue manager name.
getQueueName	Gets the name of the bridge queue.
getRemoteQName	Gets the name of the queue on MQS where the message is put after it has been transformed.

MQeMQBridgeQueue getMQQMgr

Syntax

```
public String getMQQMgr() throws Exception
```

Description

Gets the name of the MQSeries queue manager that is used to convey messages onto the MQSeries network.

Parameters

none

Return Values

A String containing the name of the MQSeries queue manager which will be used to convey queue messages onto the MQSeries network..

If the administrator specified "" or a *null* in the MQBridge queue parameter, the value returned by this string matches that returned by the **getQueueManagerName()** method.

Exceptions

java.lang.Exception If the name get fails, or if the name is invalid.

Example

```
String mqQMgrName = transformersBridgeQueue.getMQQMgr();
```

MQeMQBridgeQueue

MQeMQBridgeQueue getQueueAttribute

Syntax

```
public MQeAttribute getQueueAttribute() throws Exception
```

Description

Gets the attribute attached to the MQbridge queue.

Overrides **getQueueAttribute** in class **MQeQueue**

Parameters

none

Return values

An **MQeAttribute** object containing the attribute the bridge queue is configured with.

Exceptions

java.lang.Exception If the get fails

Example

```
MQeAttribute attribute = transformersBridgeQueue.getQueueAttribute();
```

MQeMQBridgeQueue getQueueManagerName

Syntax

```
public String getQueueManagerName() throws Exception
```

Description

Gets the name of the owning MQSeries queue manager.

Overrides **getQueueManagerName** in class **MQeQueue**.

Parameters

none

Return values

A String containing the name of the owning queue manager.

Exceptions

java.lang.Exception If the get fails or the name is invalid

Example

```
String qMgrName = transformersBridgeQueue.getQueueManagerName();
```

MQeMQBridgeQueue getQueueName

Syntax

```
public String getQueueName() throws Exception
```

Description

Gets the name of the bridge queue as it is known on MQSeries Everyplace.

Overrides **getQueueName** in class **MQeQueue** .

Parameters

none

Return values

A string containing the name of the bridge queue.

Exceptions

java.lang.Exception If the get fails or the name is invalid

Example

```
String mqQName = transformersBridgeQueue.getQueueName();
```

MQeMQBridgeQueue getRemoteQName

Syntax

```
public String getRemoteQName() throws Exception
```

Description

Gets the name of the queue on MQS that the message is put to after it has been transformed.

Parameters

none

Return values

A String containing the string contents of the remote queue name configuration information that was specified when the queue was configured by the administrator.

If the remote queue name is blank or *null*, then the value returned is the queue name of the MQSeries bridge queue itself.

Exceptions

java.lang.Exception If the get fails or the name is invalid

Example

```
String remoteQName = transformersBridgeQueue.getRemoteQName();
```

MQeMQBridgeQueueAdminMsg

Used to administer an MQBridge queue.

Package `com.ibm.mqe.mqbridge`

This class extends `MQeRemoteQueueAdminMsg`

Constants and variables

`MQeQueueAdminMsg` provides the following constants and variables :

Queue_BridgeName

Constant used when dumping this queue's details to the registry. The `MQeField` holds the name of the bridge.

```
public static final String Queue_BridgeName
```

Queue_ClientConnection

Constant used when dumping this queue's details to the registry. The `MQeField` holds the name of the client connection.

```
public static final String Queue_ClientConnection
```

Queue_MaxIdleTime

The name of the `MaxIdleTime` config parameter field that indicates how long an MQSeries bridge queue is allowed to keep an idle connection, before the connection is returned to the connection pool.

```
public static final String Queue_MaxIdleTime[]
```

Queue_MQQMgr

Constant used when dumping this queue's details to the registry. The `MQeField` holds the name of the MQS Queue Manager.

```
public static final String Queue_MQQMgr
```

Queue_RemoteQName

Constant used when dumping this queue's details to the registry. The `MQeField` holds the remote queue name.

```
public static final String Queue_MQQMgr
```

Queue_Transformer

The name of the transformer to use when converting an MQSeries Everyplace message into an MQS message.

```
public static final String Queue_Transformer
```

Queue characteristics

Name The name of the queue.

For an MQBridge queue it is the name by which the MQSeries queue is known on the MQSeries Everyplace system. (Ascii)

This characteristic is mandatory.

MQeField label: `MQeMQBridgeQueueAdminMsg.Admin_Name`

QMgrName

As described in the `com.ibm.mqe.administration.MQeQueueAdminMsg` class.

For an MQSeries bridge queue, this should hold the name of the MQSeries queue manager on which the queue is located locally, not necessarily the queue manager to which the bridge has a direct connection.

MQeField label: `MQeMQBridgeQueueAdminMsg.Queue_QMgrName`

Active**CreationDate**

As described in the com.ibm.mqe.administration.MQeQueueAdminMsg class.

MQeField label: MQeMQBridgeQueueAdminMsg.Queue_CreationDate

Description

As described in the com.ibm.mqe.administration.MQeQueueAdminMsg class.

MQeField label: MQeMQBridgeQueueAdminMsg.Queue_Description

Expiry As described in the com.ibm.mqe.administration.MQeQueueAdminMsg class.

The Message expiry time.

This value is available to the transformer of the message, so the transformer can use it to set the message expiry time of the MQSeries message that is sent to the MQSeries system.

A value of less than 1 means "never expire".

The transformer (being user customized) may choose not to use this information.

The MQSeries Everyplace message may itself contain an expiry time which over-rides this value also.

MQeField label: MQeMQBridgeQueueAdminMsg.Queue_Expiry

MaxMsgSize

As described in the com.ibm.mqe.administration.MQeQueueAdminMsg class.

The Maximum message length is an optional field. It is NOT used by the base MQBridge code at all, but is available to the rules class to decide whether the message should be sent to the MQSeries queue or not.

Checking is not performed to make sure this value is the same as that specified by the MQSeries queue this MQBridge queue refers to. For example, the rule may use this value to prevent messages over a certain length being sent to MQ, even though the MQSeries queue could accept bigger messages.

MQeField label: MQeMQBridgeQueueAdminMsg.Queue_MaxMsgSize

Mode Type

As described in the com.ibm.mqe.administration.MQeQueueAdminMsg class.

MQeField label: MQeMQBridgeQueueAdminMsg.Queue_Mode

Note: The Cryptor, Authenticator and compressor characteristics (below) define a set of queue attributes that dictate the level of security for any message being passed to this queue. From the point in MQSeries Everyplace where the message is sent initially, to the point where the message is passed to the MQBridge queue, the message is protected with at least the specified level of security, enforced by MQe. These values are NOT applicable when the MQBridge queue passes the message to the MQSeries system. The security, send and receive exits specified on the client connection configuration object

MQeMQBridgeQueueAdminMsg

on the bridge are used to protect the message as it passes to MQ. There is no checking that the link between the MQBridge queue and the MQSeries system is at least as secure as the security attributes specified in the Cryptor, Authenticator and compressor classes here.

AttrRule

As described in the com.ibm.mqe.administration.MQeQueueAdminMsg class.

MQeField label: MQeMQBridgeQueueAdminMsg.Queue_AttrRule

Authenticator

As described in the com.ibm.mqe.administration.MQeQueueAdminMsg class.

MQeField label: MQeMQBridgeQueueAdminMsg.Queue_Authenticator

Compressor

As described in the com.ibm.mqe.administration.MQeQueueAdminMsg class.

MQeField label: MQeMQBridgeQueueAdminMsg.Queue_Compressor

Cryptor

As described in the com.ibm.mqe.administration.MQeQueueAdminMsg class.

MQeField label: MQeMQBridgeQueueAdminMsg.Queue_Cryptor

TargetRegistry

As described in the com.ibm.mqe.administration.MQeQueueAdminMsg class.

MQeField label: MQeMQBridgeQueueAdminMsg.Queue_TargetRegistry

Rule

As described in the com.ibm.mqe.administration.MQeQueueAdminMsg class.

MQeField label: MQeMQBridgeQueueAdminMsg.Queue_Rule

Bridge Name

The MQBridge is the name of the MQSeries bridge that should be used to convey MQSeries Everyplace messages onto the MQSeries network. (Ascii)

This field is mandatory.

MQeField label: MQeMQBridgeQueueAdminMsg.Queue_BridgeName

MQSeries Queue Manager Proxy

The MQSeries queue manager proxy is the name of the MQSeries queue manager proxy in the MQbridge that is used to convey MQSeries Everyplace messages onto the MQSeries network.

This field is mandatory.

(This is the same as the name of the MQSeries queue manager to which the message is initially sent using a client connection).

MQeField label: MQeMQBridgeQueueAdminMsg.Queue_ClientConnection

MQSeries Remote Q name

The remote MQSeries queue name is optional.

It indicates the name of the queue to which messages are put on the MQSeries queue manager.

MQeMQBridgeQueueAdminMsg

If set to blank or *null*, it indicates that the queue name on the MQSeries queue manager (at the other end of the specified client connection) has the same name as this MQSeries bridge queue definition (so the value is derived from the "Name" field above).

This parameter provides a form of name change to allow two similarly-named queues on two MQSeries queue managers, to have different MQSeries Bridge queue names on the MQSeries Everyplace system.

MQeField label: MQeMQBridgeQueueAdminMsg.Queue_RemoteQName

Transformer class

The Transformer class is the name of the java class used to convert an MQSeries Everyplace message into an MQSeries message, before the MQSeries message is put onto the MQSeries network.

If this parameter is left blank, or is *null*, then the default transformer class for the specified MQSeries bridge (from the MQSeries Bridge field above) is used.

This parameter is optional.

MQeField label: MQeMQBridgeQueueAdminMsg.Queue_Transformer

Return idle connection timeout

The "idle time" parameter indicates the maximum number of minutes that the queue can hold an idle MQSeries connection, before it has to be released back to an idle connection pool maintained by the bridge.

If the queue is unused for a while, the underlying MQSeries connection is released back to a pool where another queue could pick it up, and use it. When the original idle queue has any message activity, it can go to the pool for another (possibly different) connection, to use.

The fact that the MQSeries connections logically underpinning an MQSeries bridge queue may be freed, and re-allocated, is transparent to users of the queue, except that a small time penalty is incurred releasing and getting an idle connection to/from the connection pool.

The idle time is in units of 1 minute.

The timer parameter is directly dependent on the granularity of the bridge heartbeat parameter.

See bridge object configuration parameters in the *MQSeries Everyplace Programming Guide*.

In the extreme case, specifying an idle time of 0 means "always free a connection to the pool once you have used it". Although this allows a very small number of MQSeries client connection channels to be effectively "shared" between a large number of MQBridge queues (provided they all use the same Bridge/MQProxy/ClientConnection details) it is at the cost of the release-to-pool and acquire-from-pool operations for each message.

A default of 5 minutes is recommended.

MQeField label: MQeMQBridgeQueueAdminMsg.Queue_MaxIdleTime

Constructor summary

Constructor	Purpose
MQeMQBridgeQueueAdminMsg	Create the administration message to administer the bridge queue.

Method summary

Method	Purpose
characteristics	Create a fields object containing the valid characteristics of a queue.

MQeMQBridgeQueueAdminMsg

Syntax

1. `public MQeMQBridgeQueueAdminMsg() throws Exception`
2. `public MQeMQBridgeQueueAdminMsg(String bridge, String mqMgrProxy, String clientConnection) throws Exception`

Description

There are two constructors:

1. This version creates and initializes a default administration message to administer the bridge queue
2. This version includes initial values for the name of the bridge, the name of the proxy, and the name of the client connection

Parameters

bridge A String containing the name of the bridge to which the administration message is directed. If set to null, or "", it is not set.

MQMgrProxy

A String containing the name of the MQS queue manager that owns the transmission queue the bridge queue is set up to read from. If set to *null* or "", it is not set.

clientConnection

A String containing the name of the client connection used to talk to the MQSeries queue manager. If set to *null* or "", it is not set.

maxIdleTimeout

The number of minutes the queue is allowed to keep hold of a connection to the MQSeries system without using it. When the queue has been unused for this duration, the connection is returned to the connection pool. If 0, the connection is returned to the connection pool immediately it has been used.

Return Values

none

Exceptions

java.lang.Exception If the message cannot be created

Example

1. MQeMQBridgeQueueAdminMsg msg;
msg = new MQeMQBridgeQueueAdminMsg();
2. MQeMQBridgeQueueAdminMsg msg;
msg = new MQeMQBridgeQueueAdminMsg("MQBridgeV100",
"lizzieQM",
"svrconn",
5);

MQeMQBridgeQueueAdminMsg characteristics

Syntax

```
public MQeFields characteristics() throws Exception
```

Description

Creates a fields object containing the valid characteristics of a queue. The **MQeFields** object contains the valid field names, it does not contain the value of each characteristic. It can be used to determine the name and type of all valid characteristics.

Overrides **characteristics** in class **MQeRemoteQueueAdminMsg** .

Parameters

none

Return values

An **MQeFields** object containing the characteristics of the queue.

Exceptions

java.lang.Exception If the **MQeFields** object cannot be created

Example

```
MQeMQBridgeQueueAdminMsg msg;  
msg = new MQeMQBridgeQueueAdminMsg( "MQBridgeV100",  
"lizzieQM",  
"svrconn",  
5 );  
MQeFields adminMsgChars = msg.characteristics();
```

MQeMQBridges

The bridges class acts as a "loader" for all the bridges. It is the class that is specified as the MQBridge class alias in the ini file and causes an instance of this class to be loaded dynamically by the MQSeries Everyplace server.

The aim of the bridges class is to load bridge objects from the registry into memory and to maintain a list of bridges that are available within the JVM. There is only ever one bridges object in a single JVM.

The server code calls the activate() method just after the constructor() and calls the close() method when the bridges should all be cleanly shut down. Between starting the bridges object and shutting it down, the server must retain a reference to the bridges object to stop it from being garbage-collected.

Package **com.ibm.mqe.mqbridge**

This class extends **MQeAdministeredObject**

Constructor summary

Constructor	Purpose
MQeMQBridges	Creates an MQBridges object

Method summary

Method	Purpose
activate	Called by the gateway/bridge server just after the constructor to activate the bridges object
close	Closes the bridges object

MQeMQBridges

Syntax

Syntax: public MQeMQBridges() throws Exception

Description

Simple constructor.

The gateway server should load this class using the MQSeries Everyplace class loader, and so the constructor is not called directly. The server should then call the activate() method passing in any configuration parameters.

Parameters

none

Return Values

none

Exceptions

java.lang.Exception

Example

```
MQeMQBridges mqBridges = (MQeMQBridges) MQe.loader.loadObject( "MQBridge" );
```

MQeMQBridges activate

Syntax

```
public void activate(com.ibm.mqe.MQeFields config) throws Exception
```

Description

Called by the MQSeries Everyplace server just after the null constructor. Used to pass configuration to the bridges object, so the bridges object can load any bridges specified in the configuration

Parameters

config	An MQeFields object that contains the complete contents of the ini file with which the MQSeries Everyplace server was initialized.
---------------	--

```
Fields=Aliases
...
(ascii)MQBridge=com.ibm.mqe.mqbridge.MQeMQBridges
Fields=ChannelManager
...
Fields=Listener
...
Fields=QueueManager
...
Fields=MQBridge
(ascii)LoadBridgeRule=RuleClass
```

Return values

none

Exceptions

java.lang.Exception	Fails if the underlying server could not activate the bridges object, or the configuration parameters supplied were invalid.
----------------------------	--

Example

```
mqBridges.activate( myConfigFields );
```

MQeMQBridges close

Syntax

```
public void close() throws Exception
```

Description

Closes the bridges objects as cleanly as possible.

Parameters

none

Return values

none

Exceptions

java.lang.Exception	If the objects could not be shut down cleanly.
----------------------------	--

Example

MQeMQBridges

```
mqBridges.close( );
```

MQeMQBridgesAdminMsg

Used to encapsulate an administration command that acts on the MQeBridges object.

This message is created by the application doing the administration.

The logic performed on the target MQSeries Everyplace system is also in this class.

The administration queue invokes the performAction method.

Package **com.ibm.mqe.mqbridge**

This class extends **MQeAdminMsg**

Constants and variables

MQeQueueAdminMsg provides the following constants and variables in addition to those provided by **MQeAdminMsg**:

Action_Start

Operation code for starting an administered object

```
public static final int Action_Start
```

Action_Stop

Operation code for stopping an administered object.

```
public static final int Action_Stop
```

Constructor summary

Constructor	Purpose
MQeMQBridgesAdminMsg	Creates and initializes an MQeBridgesAdminMsg object.

Method summary

Method	Purpose
characteristics	Creates an MQSeries Everyplace fields object containing all the MQe fields required for an administration message of this type.
getName	Gets the name of the administered object to be created.
start	Tells the administered object to start.
stop	Tells the administered object to stop

MQeMQBridgesAdminMsg

Syntax

1. `public MQeMQBridgesAdminMsg() throws Exception`
2. `public MQeMQBridgesAdminMsg(boolean affectChildren) throws Exception`

Description

There are two constructors:

1. This version creates and initializes a default MQeMQBridgesAdminMsg
2. This version includes a flag to determine whether children should be affected by the administration commands

Parameters

MQeMQBridgesAdminMsg

affectChildren A boolean flag indicating whether or not this administration message affects the children of the bridges object.

true means it is allowed to affect the children, **false** indicates that the children should not be affected.

Some commands ALWAYS affect the children (Stop for example).

This flag is sometimes transferred into the **MQeEvent** that is sent to the bridges administered object.

Return Values

none

Exceptions

java.lang.Exception If the **MQeField** holding the **affectChildren** flag cannot be created.

Example

```
MQeMQBridgesAdminMsg msg = new MQeMQBridgesAdminMsg(true);
```

Related functions

MQeEvent

MQeMQBridgesAdminMsg characteristics

Syntax

```
public MQeFields characteristics() throws Exception
```

Description

Creates an MQSeries Everyplace fields object containing all the MQSeries Everyplace fields required for an administration message of this type.

Overrides **characteristics** in class **MQeBridgesAdminMsg** .

Parameters

none

Return values

An **MQeFields** object containing the characteristics of the resource.

Exceptions

java.lang.Exception If the **MQeFields** object cannot be created

Example

```
MQeMQBridgesAdminMsg msg = new MQeMQBridgesAdminMsg(true);  
MQeFields theseCharacteristics = msg.characteristics();
```

MQeMQBridgesAdminMsg getName

Syntax

```
public String getName()
```

Description

Returns the name of the administered object we want to create.

Overrides **getName** in class **MQeAdminMsg**

Parameters

none

Return values*null* if the name was not set, otherwise a string.**Exceptions**

none

Example

```
MQeMQBridgesAdminMsg msg = new MQeMQBridgesAdminMsg(true);
String name = msg.getName();
```

MQeMQBridgesAdminMsg start**Syntax**

1. `public void start() throws Exception`
2. `public void start(boolean affectChildren) throws Exception`
3. `public void start(MQeFields fields) throws Exception`

Description

There are three versions:

1.

This version is used by the source of the administration message and tells the administered object to start. It also tells all its child objects to start.

2.

This version is used by the source of the administration message and tells the administered object to start. It also tells all its child objects to start depending on the value of a parameter.

3.

This version tells the administered object to start and also tells all its child objects to start or not, depending on the value of the **affectChildren** field.

Other bridge related parameters can also be passed in the **MQeFields** object.

Parameters

affectChildren A boolean that indicates whether the command operation set in this administration message is allowed to affect the children of the bridges object.

true means it is allowed to affect the children, **false** indicates that the children should not be affected.

Some commands ALWAYS affect the children (stop for example).

This flag is sometimes transferred into the **MQeEvent** that is sent to the bridges administered object.

fields

An **MQeFields** object that contains a set of fields that can contain the bridge name, the **affectChildren** flag, and other bridge related parameters.

MQeMQBridgesAdminMsg

These fields are blindly copied into this administration message ready for transmission. No validation is performed on these field parameters.

If the **affectChildren** flag is present in the fields parameter, it overrides the default value of **true**.

Return values

none

Exceptions

java.lang.Exception

If the start action or the **affectChildren** flag, or any of the passed fields cannot be set into MQeFields in this administration message

Example

1. MQeMQBridgesAdminMsg msg = new MQeMQBridgesAdminMsg();
msg.start();
2. MQeMQBridgesAdminMsg msg = new MQeMQBridgesAdminMsg();
msg.start(true);
3. MQeMQBridgesAdminMsg msg = new MQeMQBridgesAdminMsg();
MQeFields fields = new MQeFields();
fields.putBoolean(MQeCharacteristicLabels.MQE_FIELD_LABEL_AFFECT_CHILDREN,
true);
msg.start(fields);

MQeMQBridgesAdminMsg stop

Syntax

1. public void stop() throws Exception
2. public void stop(MQeFields fields) throws Exception

Description

There are two versions:

1.

This version is used by the source of the administration message and tells the administered object to stop. It also tells all its child objects to stop.

2.

This version tells the administered object to stop and also tells all its child objects to stop.

This version accepts a set of fields that can contain the fields that identify which **MQeAdministeredObject** should be stopped.

Parameters

fields

An **MQeFields** object. These fields are copied into the administration message fields, overriding any current field values.

Return values

none

Exceptions

java.lang.Exception

If the action cannot be set into this administration message

Example

MQeMQBridgesAdminMsg

1. `MQeMQBridgesAdminMsg msg = new MQeMQBridgesAdminMsg();
msg.start();`
2. `MQeMQBridgesAdminMsg msg = new MQeMQBridgesAdminMsg();
MQeFields fields = new MQeFields();
fields.putBoolean(MQeCharacteristicLabels.MQE_FIELD_LABEL_AFFECT_CHILDREN,
true);
msg.start(fields);`

MQeMQMGrProxyAdminMsg

This message is created by the application doing the administration and is used to encapsulate an administration command that acts on the MQeMQMGrProxy object.

The logic performed on the target MQSeries Everyplace system is also in this class.

The administration queue invokes the performAction method.

Package **com.ibm.mqe.mqbridge**

This class extends **MQeMQBridgeAdminMsg**

Constructor summary

Constructor	Purpose
MQeMQMGrProxyAdminMsg	Creates and initializes an MQeMQMGrProxyAdminMsg object.

Method summary

Method	Purpose
characteristics	Creates an MQSeries Everyplace fields object containing all the MQe fields required for an administration message of this type.
getMQMGrProxyName	Gets the MQS queue manager proxy name from the administered object.
getName	Gets the name of the administered object to be created.
putMQMGrProxyName	Puts the MQS queue manager proxy name in an MQSeries Everyplace field in the MQSeries Everyplace fields administration message object
setName	Puts the name information in an MQSeries Everyplace field in the MQe fields administration message object

MQeMQMGrProxyAdminMsg

Syntax

1. public MQeMQMGrProxyAdminMsg() throws Exception
2. public MQeMQMGrProxyAdminMsg(String bridge, String MQMGrProxy, boolean affectChildren) throws Exception

Description

There are two constructors:

1. This version creates and initializes a default MQeMQMGrProxyAdminMsg
2. This version includes the MQSeries Everyplace queue manager name, the name of the bridge, and the name of the proxy

Parameters

bridge A String containing the name of the bridge to which the administration message is directed. If set to *null*, or "", it is not set.

MQMGrProxy

A String containing the name of the proxy to which the administration message is directed. If set to *null*, or "", it is not set.

affectChildren A boolean flag indicating whether or not this administration message affects the children of the bridges object.

true means it is allowed to affect the children, **false** indicates that the children should not be affected.

Return Values

none

Exceptions

java.lang.Exception If any fields could not be set or if the MQSeries queue manager name is invalid.

Example

```
1. MQeMQMGrProxyAdminMsg msg = new MQeMQMGrProxyAdminMsg();
2. MQeMQMGrProxyAdminMsg msg;
   msg = new MQeMQMGrProxyAdminMsg("MQBridgeV100", "lizzieQM");
```

MQeMQMGrProxyAdminMsg characteristics**Syntax**

```
public MQeFields characteristics() throws Exception
```

Description

Creates an MQSeries Everyplace fields object containing all the MQSeries Everyplace fields required for an administration message of this type.

Returns a fields object containing the characteristics of the resource. The complete set of field names and types for the resource can be determined from the resulting fields object.

Overrides **characteristics** in class **MQeBridgeAdminMsg** .

Parameters

none

Return values

An MQeFields object containing the characteristics of the resource.

Exceptions

java.lang.Exception If the **MQeFields** object cannot be created

Example

```
MQeMQMGrProxyAdminMsg msg;
msg = new MQeMQMGrProxyAdminMsg("MQBridgeV100", "lizzieQM");
MQeFields proxyChars = msg.characteristics();
```

MQeMQMGrProxyAdminMsg getMQMGrProxyName**Syntax**

```
public String getMQMGrProxyName() throws Exception
```

Description

Gets the MQSeries queue manager proxy name from the administration object.

MQeMQMGrProxyAdminMsg

This method also checks the field for validity. It can be issued against an MQeMQMGrProxyAdminMsg or one of its descendants.

Parameters

none

Return values

none

Exceptions

java.lang.Exception If the get fails

Example

```
MQeMQMGrProxyAdminMsg msg;  
msg = new MQeMQMGrProxyAdminMsg("MQBridgeV100", "lizzieQM");  
String proxyName = msg.getMQMGrProxyName();
```

MQeMQMGrProxyAdminMsg getName

Syntax

```
public String getName()
```

Description

Returns the name of the current administration object. When issued against an object of this class it is identical to getMQMGrProxyName().

Overrides **getName** in class **MQeMQBridgeAdminMsg**

Parameters

none

Return values

null if the name was not set, a string otherwise.

Exceptions

none

Example

```
MQeMQMGrProxyAdminMsg msg;  
msg = new MQeMQMGrProxyAdminMsg("MQBridgeV100", "lizzieQM");  
String proxyName = msg.getMName();
```

MQeMQMGrProxyAdminMsg putMQMGrProxyName

Syntax

```
public void putMQMGrProxyName(String mqMGrProxy) throws Exception
```

Description

Puts the MQSeries queue manager proxy name in an MQSeries Everyplace field in the MQSeries Everyplace fields administration message object.

Parameters

mqMGrProxy

A String containing the name of the proxy to which the administration message is directed. If set to *null*, or "", it is not set.

Return values

none

Exceptions

MQeMQMGrProxyAdminMsg

java.lang.Exception

If any fields could not be set or if the MQSeries queue manager name is invalid

Example

```
MQeMQMGrProxyAdminMsg msg = new MQeMQMGrProxyAdminMsg();  
msg.putMQMGrProxyName("lizzieQM");
```

MQeMQMGrProxyAdminMsg setName

Syntax

```
public void setName(String bridge,  
                    String mqMGrProxy) throws Exception
```

Description

Puts the bridge name and proxy name into the administration message object.

Parameters

bridge A String containing the name of the bridge to which the administration message is directed. If set to *null*, or "", it is not set.

mqMGrProxy A String containing the name of the proxy to which the administration message is directed. If set to *null*, or "", it is not set.

Return values

none

Exceptions

java.lang.Exception

If any fields could not be set or if the MQSeries queue manager name is invalid.

Example

```
MQeMQMGrProxyAdminMsg msg = new MQeMQMGrProxyAdminMsg();  
msg.setName("MQBridgeV100", "lizzieQM");
```

MQeRunState

A class that holds the "run state" of an administered object, for example whether it's running, stopped, quiescing, or starting.

Package **com.ibm.mqe.mqbridge**

This class extends **MQeBridgeServices**

Constants and variables

MQeQueueAdminMsg provides the following constants and variables in addition to those supplied by **MQeBridgeServices**:

RUN_STATE_RUNNING

Administered objects have this state when they are active.

```
public static final int RUN_STATE_RUNNING
```

RUN_STATE_STOPPED

Administered objects have this state when they are inactive.

```
public static final int RUN_STATE_STOPPED
```

MQeTransformerInterface

All classes that transform MQMessages to MQeMsgObjects, and vice versa, must conform to this interface.

Package **com.ibm.mqe.mqbridge**

Method summary

Method	Purpose
activate	Tells the class implementing this interface any parameters that may have been supplied with the transformer definition
transform	Converts the given MQMessage into an MQeMsgObject and vice versa.

MQeTransformerInterface activate

Syntax

```
public abstract void activate(StringTokenizer params) throws Exception
```

Description

Tells the class implementing this interface any parameters that may have been supplied with the transformer definition

Parameters

params StringTokenizer containing the transformer definition parameters.

Return values

none

Exceptions

java.lang.Exception If the parameters are not correct, valid, or if there is a problem initializing the transformer.

MQeTransformerInterface transform

Syntax

1. `public abstract MQeMsgObject transform(MQMessage msg, String remoteQMGrName, String remoteQName) throws Exception`
2. `public abstract MQMessage transform(MQeMsgObject msg, MQeMQBridgeQueue queue, MQPutMessageOptions pmo) throws Exception`

Description

There are two versions of this method:

1. This version converts the given MQMsg into an MQeMessageObject
2. This version converts the given MQeMsgObject into an MQMessage

Parameters

msg The MQSeries or MQSeries Everyplace message that is to be transformed.

MQeTransformerInterface

remoteQMgrName	The name of the destination MQSeries Everyplace queue manager (obtained from the remote queue definition on MQSeries).
remoteQName	The name of the destination MQSeries Everyplace queue (obtained from the remote queue definition on MQSeries).
msg	The MQSeries Everyplace message that is to be transformed.
queue	A reference to the bridge queue that accepted the message
remoteQName	The name of the destination MQSeries Everyplace queue (obtained from the remote queue definition on MQSeries).
pmo	A reference to a blank MQPutMessageOptions object that can be modified by the transformer. This parameter enables the user to specify any context options that are needed to put the new MQSeries Everyplace message to MQSeries.

Return values

1. The transformed message in the form of an **MQeMsgObject**
2. The transformed message in the form of an **MQMessage**

Exceptions

java.lang.Exception	If any of the parameters are invalid, if the message is in a format that this transformer does not understand, or if there is a problem when the transformation is taking place.
----------------------------	--

Example

Chapter 9. Classes in com.ibm.mqe.adapters

This section contains detailed information about the following MQSeries
Everyplace classes:

Table 19. Classes in com.ibm.mqe.adapters

Class name	Purpose
MQeDiskFieldsAdapter	Provides support for reading and writing MQeFields information to a local disk.
MQeMemoryFieldsAdapter	Provides a non-persistent store for MQeFields information
MQeTcpipAdapter	Provides support for reading and writing data over TCP/IP streams.
MQeTcpipHttpAdapter	Extends the MQeTcpipAdapter to provide basic support for the HTTP 1.0 protocol.
MQeTcpipLengthAdapter	Extends the MQeTcpipAdapter to provide a simple, byte efficient protocol.
MQeTcpipHistoryAdapter	Extends the MQeTcpipLengthAdapter to provide a more efficient protocol that caches recently used data.

MQeDiskFieldsAdapter

MQeDiskFieldsAdapter

The MQeDiskFieldsAdapter is an adapter that provides support for reading and writing MQeFields information to a local disk. A single instance of this class refers to a single file on disk.

Package **com.ibm.mqe.adapter**

This class is a descendant of **MQeAdapter**

Method summary

Method	Purpose
accept	Checks the extension of the given string to see if it ends with the currently defined file filter.
activate	Initializes a new MQeDiskFieldsAdapter object.
close	Closes the current MQeDiskFieldsAdapter object.
control	Allows the user to set the file filter or to list the files in the current file's folder.
erase	Removes a file from disk.
open	Opens a file for use.
readObject	Reads an MQeFields object from the current file.
status	Provides information on the currently active settings.
writeObject	Writes an MQeFields object to disk.

MQeDiskFieldsAdapter accept

Syntax

```
public boolean accept( File dir, String name )
```

Description

Checks the extension of the given string to see if it ends with the currently defined file filter.

Parameters

File dir Not used

String name The name of the file to be checked

Return Values

True If the named parameter ends with the current filter String.

Exceptions

None

Example

```
Boolean isText = mydfa.accept(null, ".txt");
```

MQeDiskFieldsAdapter activate

Syntax

```

public void activate( String fileDesc,
                    Object param,
                    Object options,
                    int lrecl,
                    int noRec ) throws Exception

```

Description

This method initializes a new MQeDiskFieldsAdapter object. Sets the Quality of Service (QOS) Size parameter to be the size of the file or MQeFields object.

Parameters**String fileDesc**

The directory path to be opened. Only the path to a file should be specified here. If you want to open a file the name of the file is specified on the open. This value is used in a number of subsequent methods as the *File descriptor*.

Object options

A String that specifies the open options:

MQe.MQe_Adapter_READ

Reads the file

MQe.MQe_Adapter_WRITE

Writes the file

MQe.MQe_Adapter_UPDATE

Reads and writes the file

param Not used

lrecl Not used

noec Not used

Return Values

None

Exceptions**MQeException**

If the specified directory hierarchy could not be created dynamically.

Example

```
mydfa.activate("./mydir/mysubdir", null, MQe.MQeAdapter_READ, 0, 0);
```

MQeDiskFieldsAdapter close**Syntax**

```
public void close( Object opt ) throws Exception
```

Description

Closes the current adapter so that it can no longer be read from or written to.

Parameters

None

Return Values

None

Exceptions

None

MQeDiskFieldsAdapter

Example

```
mydfa.close(null);
```

MQeDiskFieldsAdapter control

Syntax

```
public Object control( Object opt, Object ctrlObj ) throws Exception
```

Description

Allows the user to set the file filter or to list the files in the current directory.

Parameters

Object opt A String containing one of the following values:
MQe_Adapter_FILTER
Sets the filter to the value of the second parameter

MQe_Adapter_LIST
Lists the files

Object ctrlObj

A String that specifies the filter to use (if MQe_Adapter_FILTER is specified). Not used for MQe_Adapter_LIST

Return Values

null for MQe_Adapter_FILTER, or a String[] list of files in the case of MQe_Adapter_LIST.

Exceptions

None

Example

```
String[] fileList = (String[]) mydfa.control(MQe.MQe_Adapter_LIST,  
null);
```

MQeDiskFieldsAdapter erase

Syntax

```
public void erase( Object opt ) throws Exception
```

Description

Erases a file from disk. Does not erase directories.

Parameters

Object opt A String that contains the file name to be deleted. Before the name is passed to the operating system's delete routine, the file name is appended to the current value of the *File descriptor*.

Return Values

None

Exceptions

MQeException

If the delete failed, or if the argument supplied referred to a directory or was not a String.

Example

```
mydfa.erase("myfile.txt");
```

MQeDiskFieldsAdapter open

Syntax

```
public void open( Object opt ) throws Exception
```

Description

Closes any open files and then opens a file for use.

Parameters

Object opt A String that specifies the filename to be opened. This file should exist in the folder specified on the *File Descriptor* value (supplied on the activate call).

Return Values

None

Exceptions

None

Example

```
mydfa.open("myfile.txt");
```

MQeDiskFieldsAdapter readObject

Syntax

```
public synchronized Object readObject( Object opt ) throws Exception {
```

Description

Reads an MQeFields object from the current file.

Parameters

Object opt An MQeFields object that can be used to provide a filter within the directory object. Otherwise *null*.

Return Values

The MQeFields object that was read.

Exceptions

MQeException

If no file matching the filter was found, or if the file is not open for reading.

Example

```
MQeFields myFields = (MQeFields) mydfa.readObject(null);
```

MQeDiskFieldsAdapter status

Syntax

```
public String status( Object opt ) throws Exception
```

Description

Provides details of the current settings.

Parameters

Object opt A String containing one of the following values:

MQe.MQe_Adapter_FILTER

Gets the name of the currently active filter

MQe.MQe_Adapter_FILENAME

Gets the name of the currently active file.

MQeDiskFieldsAdapter

Return Values

A String containing either the name of the filter or the name of the file.

Exceptions

None

Example

```
String filename = (String) mydfa.status(MQe.MQe_Adapter_FILENAME);
```

MQeDiskFieldsAdapter writeObject

Syntax

```
public synchronized void writeObject(Object opt, Object data)  
    throws Exception
```

Description

Writes an MQeFields object to disk.

Parameters

Object opt Not used

Object data An MQeFields object to dump to disk

Return Values

None

Exceptions

MQeException

If an invalid data argument was supplied, or if the file was not opened for writing.

Example

```
mydfa.writeObject(null, myMQeFields);
```

MQeMemoryFieldsAdapter

The MQeMemoryFieldsAdapter is an adapter that provides support for reading and writing MQeFields information to memory. This adapter behaves in a similar fashion to the MQeDiskFieldsAdapter, except that the directory structure is actually an internal hash table.

Package **com.ibm.mqe.adapter**

This class is a descendant of **MQeAdapter**

Method summary

Method	Purpose
activate	Initializes a new MQeMemoryFieldsAdapter object.
close	Closes the current MQeMemoryFieldsAdapter object.
control	Allows the user to list the files in the current file's folder.
erase	Removes a file from memory.
open	Opens a file for use.
readObject	Reads an MQeFields object from the current file.
status	Provides information on the current settings.
writeObject	Writes an MQeFields object to memory.

MQeMemoryFieldsAdapter activate

Syntax

```
public void activate( String fileDesc,
                    Object param,
                    Object options,
                    int lrecl,
                    int noRec ) throws Exception
```

Description

This method initializes a new MQeMemoryFieldsAdapter object.

Parameters

String fileDesc

The directory path to be opened. Only the path to a file should be specified here. If you want to open a file the name of the file is specified on the open. This value is used in a number of subsequent methods as the *File descriptor*.

Object options

A String that specifies the open options:

MQe.MQe_Adapter_READ

Reads the file

MQe.MQe_Adapter_WRITE

Writes the file

MQe.MQe_Adapter_UPDATE

Reads and writes the file

param Not used

lrecl Not used


```
String[]] fileList = (String[]]) mydfa.control(MQe.MQe_Adapter_LIST,
null);
```

MQeMemoryFieldsAdapter erase

Syntax

```
public void erase( Object opt ) throws Exception
```

Description

Erases a file from memory. Does not erase directories.

Parameters

Object opt A String that contains the file name to be deleted. Before the name is passed to the hash tables remove routine, the file name is appended to the current value of the *File descriptor*.

Return Values

None

Exceptions

MQeException

If the argument supplied referred to a directory or was not a String.

Example

```
mydfa.erase("myfile.txt");
```

MQeMemoryFieldsAdapter open

Syntax

```
public void open( Object opt ) throws Exception
```

Description

Closes any open files and then opens a file for use.

Parameters

Object opt A String that specifies the filename to be opened. This file should exist in the folder specified on the *File Descriptor* value (supplied on the activate call).

Return Values

None

Exceptions

None

Example

```
mymfa.open("myfile.txt");
```

MQeMemoryFieldsAdapter readObject

Syntax

```
public synchronized Object readObject( Object opt ) throws Exception
```

Description

Reads an MQeFields object from the current file.

Parameters

MQeTcpipAdapter

The MQeTcpipAdapter is an adapter that provides support for reading and writing data over TCP/IP streams.

This adapter cannot be used to connect two MQSeries Everyplace applications, as it does not provide a protocol for the exchange of data. Various adapters that inherit from this class provide some possible protocols . MQeTcpipLengthAdapter, and MQeTcpipHttpAdapter, for example.

An instance of this class can be used to either listen for incoming TCP/IP connections, or connect to an existing listener. It is not possible for a single MQeTcpipAdapter object to do both.

Package **com.ibm.mqe.adapter**

This class is a descendant of **MQeAdapter**

Method summary

Method	Purpose
activate	Initializes a new MQeTcpipAdapter object.
close	Closes the current MQeTcpipAdapter object.
control	Sets or gets information on the current adapter instance.
open	Opens a socket for use.
read	Reads bytes from the underlying TCP/IP stream.
readln	Reads a line of data from the underlying TCP/IP stream.
status	Returns information on the adapter.
write	Writes bytes to the underlying TCP/IP stream.
writeln	Writes a line of data to the underlying TCP/IP stream.

MQeTcpipAdapter activate

Syntax

```
public void activate( String fileDesc,
                    Object param,
                    Object options,
                    int lrecl,
                    int noRec ) throws Exception
```

Description

This method initializes a new MQeTcpipAdapter object. While the MQeTcpipAdapter is not intended for use by MQSeries Everyplace applications, this method should be called inside the activate method of all the adapters that extend this class.

Parameters

String fileDesc

The String that represents this adapter. ("Network:1.2.3.4:8081" for example.)

Object param The parameters is given to the created adapter when an ACCEPT is issued.

MQeTcpipAdapter

Object options

A String that specifies the open options:

MQe.MQe_Adapter_LISTEN

Creates a listening socket

MQe.MQe_Adapter_ACCEPT

Accepts a connection

MQe.MQe_Adapter_LOCALHOST

Returns the local host address

MQe.MQe_Adapter_NETWORK

Returns the network type (TCPIP)

MQe.MQe_Adapter_PERSIST

Copies the persistent object on the accept call

MQe.MQe_Adapter_QOSINPUTS

Returns valid Quality of Service (QOS) names

MQe.MQe_Adapter_SETSOCKET

Reserved

For example, to configure a TCP/IP-based adapter to perform listening operations, specify MQe.MQe_Adapter_LISTEN for this parameter.

int lrecl

Specifies the TCP/IP block size. If the value supplied is less than zero the default of 4096 is used.

int norec

The time-out value of the socket in milliseconds.

Return Values

None

Exceptions

None

Example

```
super.activate(fileDesc, param, options, lrecl,  
noRec);
```

MQeTcpipAdapter close

Syntax

```
public void close( Object opt ) throws Exception
```

Description

Closes the current adapter so that it can no longer be read from or written to.

Parameters

Object opt

A String that contains MQe.MQe_Adapter_FINAL, if this is the last close for this adapter. If the adapter was opened with the MQe.MQe_Adapter_PERSIST option set, close has no effect unless the MQe.MQe_Adapter_FINAL option is included on the close method call.

Return Values

None

Exceptions

None

Example

```
super.close(null);
```

MQeTcpipAdapter control**Syntax**

```
public Object control( Object opt, Object ctrlObj ) throws Exception
```

Description

Sets or gets information on the current adapter instance.

Parameters

Object opt An object, supplied as a String that may contain any of the following:

MQe.MQe_Adapter_ACCEPT

Waits for and accepts a connection (listening sockets)

MQe.MQe_Adapter_SETSOCKET

Uses the supplied socket object

MQe.MQe_Adapter_PULSE

Starts a timer pulse thread

MQe.MQe_Adapter_QOSINPUTS -

Returns a list of Quality of Service (QOS) input parameters

;

Object ctrlObj

For MQe.MQe_Adapter_ACCEPT, this is a String representation of the *file descriptor* of the connecting adapter ("HTTP:1.2.3.4..." for example). For MQe.MQe_Adapter_SETSOCKET, this is the socket that should be used.

Return Values

For MQe.MQe_Adapter_ACCEPT, this is the adapter object that was created as a result of a connection request. Otherwise *null*, or an object that wrappers the information that was requested.

Exceptions**MQeException**

If the parameter object was not correct for the given option.

Example

```
super.control(opt, ctrlObj);
```

MQeTcpipAdapter open**Syntax**

```
public void open( Object opt ) throws Exception
```

Description

Opens a socket for use. If this is a listening adapter then a new server socket is created. Otherwise a new socket is created.

MQeTcpipAdapter

Parameters

Object opt Not used

Return Values

None

Exceptions

None

Example

```
super.open(null);
```

MQeTcpipAdapter read

Syntax

```
public byte[] read( Object opt, int recordSize )  
throws Exception
```

Description

Reads bytes from the underlying TCP/IP stream

Parameters

Object opt Not used

int recordSize The maximum buffer size to read in one go. If this parameter is less than or equal to zero, then the Size parameter from the Quality of Service (QOS) object is used as a default.

Return Values

The data in a byte array of up to the number of bytes specified. The block may be an array of less than the requested number of bytes.

Exceptions

EOFException

If no more data is available.

Example

```
byte[] results = super.read(null, 4096);
```

MQeTcpipAdapter readln

Syntax

```
public String readln( Object opt ) throws Exception
```

Description

Reads a line of text from the underlying TCP/IP stream

Parameters

Object opt Not used

Return Values

A String containing the line that was read.

Exceptions

EOFException

If no more data is available.

Example

```
String results = super.readln(null);
```

MQeTcpipAdapter status

Syntax

```
public String status( Object opt ) throws Exception
```

Description

Provides details of the current settings.

Parameters

Object opt A string containing one of the following:

MQe.MQe_Adapter_NETWORK

Gets the type of network in use

MQe.MQe_Adapter_LOCALHOST

Gets the name of the local host

Return Values

The literal 'TCPIP' for MQe.MQe_Adapter_NETWORK or the name of the local machine for MQe.MQe_Adapter_LOCALHOST.

Exceptions

None

Example

```
String netType = (String) super.status(MQe.MQe_Adapter_NETWORK);
```

MQeTcpipAdapter write

Syntax

```
public void write( Object opt, int recordSize, byte data[] )
                throws Exception
```

Description

Writes the given array of bytes to the TCP/IP output stream.

Parameters

Object opt A string containing the options to be used on the write. Currently only MQe.MQe_Adapter_FLUSH is supported. This means flush the data currently in the buffers.

int recordSize The number of bytes to write.

byte[] data The data to be written.

Return Values

None

Exceptions

None

Example

```
super.write(MQe.MQe_Adapter_FLUSH, 4096, myByteArray);
```

MQeTcpipAdapter writeln

Syntax

```
public void writeln( Object opt, String data ) throws Exception
```

Description

Writes the given String to the output stream, followed by a new-line character.

MQeTcpipAdapter

Parameters

Object opt A String containing the options to be used on the write. Currently only MQe.MQe_Adapter_FLUSH is supported. This means flush the data currently in the buffers.

String data The data to be written.

Return Values

None

Exceptions

None

Example

```
super.writeln(null, "Hello");
```


MQeTcpipHttpAdapter

The MQeTcpipHttpAdapter adds support for the HTTP 1.0 protocol to the MQeTcpipAdapter. This enables two MQSeries Everyplace systems to communicate over a TCP/IP network.

This adapter provides read and write calls that attach HTTP headers to data flows on the TCP/IP stream. As HTTP flows are generally permitted through certain ports on a firewall, this allows MQSeries Everyplace data to flow through a firewall and allows access to the World Wide Web.

Package **com.ibm.mqe.adapter**

This class is a descendant of **MQeTcpipAdapter**. All usual adapter operations that are not listed here use the implementation provided by this super-class.

Method summary

Method	Purpose
read	Reads bytes from the underlying TCP/IP stream.
readEOF	Reads bytes from the underlying TCP/IP stream, until the end of file marker.
readln	Reads a line of data from the underlying TCP/IP stream.
write	Writes bytes to the underlying TCP/IP stream.
writeln	Writes a line of data to the underlying TCP/IP stream.

MQeTcpipHttpAdapter read

Syntax

```
public byte[] read( Object opt, int recordSize )
                 throws Exception
```

Description

Reads bytes from the underlying TCP/IP stream

Parameters

Object opt One of the following:

- MQe.MQe_Adapter_HEADER**
Reads the HTTP header information
- MQe.MQe_Adapter_CONTENT**
Reads the data content

int recordSize The maximum buffer size to read in one read operation.

Return Values

The data in a byte array of up to the number of bytes specified. The block may be an array of less than the requested number of bytes.

Exceptions

EOFException
If no more data is available.

Example

MQeTcpipHttpAdapter

```
|         byte[] results = myHttpAdapter.read  
|         (MQe.MQe_Adapter_HEADER + MQe.MQe_Adapter_CONTENT,  
|         4096);
```

MQeTcpipHttpAdapter readEOF

Syntax

```
|         public byte[] readEOF( Object opt ) throws Exception
```

Description

Reads bytes from the underlying TCP/IP stream, until the end of file marker. Equivalent to MQeTcpipHttpAdapter.read(opt, -1)

Parameters

Object opt One of the following:

MQe.MQe_Adapter_HEADER

Reads the HTTP header information

MQe.MQe_Adapter_CONTENT

Reads the data content

Return Values

The data in a byte array of up to the number of bytes specified. The block may be an array of less than the requested number of bytes.

Exceptions

EOFException

If no more data is available.

Example

```
|         byte[] results = myHttpAdapter.readEOF(MQe.MQe_Adapter_HEADER);
```

MQeTcpipHttpAdapter readLn

Syntax

```
|         public byte[] readLn( Object opt ) throws Exception {
```

Description

Reads a line from the underlying TCP/IP stream

Parameters

Object opt One of the following:

MQe.MQe_Adapter_HEADER

Returns HTTP header information

NULL.

Returns the next line from the input stream

Return Values

A String containing the line that was read.

Exceptions

EOFException

If no more data is available.

Example

```
|         String top = myHttpAdapter.readLn(MQe.MQe_Adapter_HEADER);
```

MQeTcpipHttpAdapter write

Syntax

```
public void write( Object opt, int recordSize, byte data[] )
                throws Exception
```

Description

Writes the given array of bytes to the TCP/IP output stream. Once written, the data stream is flushed to prevent unwanted data being left on the sender.

Parameters

Object opt One of the following:

MQe.MQe_Adapter_HEADER

Tells the adapter to attach header information (as a POST request)

MQe.MQe_Adapter_HEADERSP

Tells the adapter to attach an HTTP response to the data (200 OK)

int recordSize The number of bytes to write.

byte[] data The data to be written.

Return Values

None

Exceptions

None

Example

```
myHttpAdapter.write(MQe.MQe_Adapter_HEADER, 4096,
                    myByteArray);
```

MQeTcpipHttpAdapter writeln

Syntax

```
public void writeln( Object opt, String data ) throws Exception
```

Description

Writes the given String to the output stream. Once written, the data stream is flushed to prevent unwanted data being left on the sender.

Parameters

Object opt One of the following:

MQe.MQe_Adapter_HEADER

Tells the adapter to attach header information (as a GET request)

MQe.MQe_Adapter_HEADERSP

Tells the adapter to attach an HTTP response to the data (200 OK)

String data The data to be written.

Return Values

None

Exceptions

None

MQeTcpipHttpAdapter

Example

```
|  
|         myHttpAdapter.writeln(MQe.MQe_Adapter_HEADER,  
|                               "Hello");  
|  
|
```

MQeTcpipLengthAdapter

The MQeTcpipLengthAdapter adds a simple protocol to the MQeTcpipAdapter that enables two MQSeries Everyplace systems to communicate over a TCP/IP network.

This adapter overrides the read and write calls from MQeTcpipAdapter to attach a header of between one and three bytes to each data flow. This header indicates the length of the following data. Using this protocol a flow of up to 1,073,741,823 bytes (230 – 1) is possible.

Package **com.ibm.mqe.adapter**

This class is a descendant of **MQeTcpipAdapter**. All usual adapter operations that are not listed here use the implementation provided by this super-class.

Method summary

Method	Purpose
read	Reads bytes from the underlying TCP/IP stream.
write	Writes bytes to the underlying TCP/IP stream.

MQeTcpipLengthAdapter read

Syntax

```
public byte[] read( Object opt, int recordSize )
    throws Exception {
```

Description

Reads bytes from the underlying TCP/IP stream

Parameters

Object opt

MQe.MQe_Adapter_HEADER

Reads the header (data length) information

int recordSize The maximum buffer size to read in one read operation.

Return Values

The data in a byte array of up to the number of bytes specified. The block may be an array of less than the requested number of bytes.

Exceptions

EOFException

If no more data is available.

Example

```
byte[] results = myLengthAdapter.read(MQe.MQe_Adapter_HEADER, 4096);
```

MQeTcpipLengthAdapter write

Syntax

```
public void write( Object opt, int recordSize, byte data[] )
    throws Exception
```

MQeTcpiLengthAdapter

Description

Writes the given array of bytes to the TCP/IP output stream. Once written, the data stream is flushed to prevent unwanted data being left on the sender.

Parameters

Object opt

MQe.MQe_Adapter_HEADER

A String that sets the header (data length) information

int recordSize The number of bytes to write.

byte[] data The data to be written.

Return Values

None

Exceptions

None

Example

```
myLengthAdapter.write(MQe.MQe_Adapter_HEADER, 4096, myByteArray);
```

MQeTcpipHistoryAdapter

The MQeTcpipHistoryAdapter provides an extra protocol layer on top of MQeTcpipLengthAdapter that supports persistent socket connections and the ability to encode recently used data in a shortened form. This makes the MQeTcpipHistoryAdapter a highly efficient communications mechanism in terms of general packet size.

Care needs to be exercised if channel level encryption or compression is used with the history encoding option, as it is possible for the sending and receiving adapters to contain different history data, (if the encryption keys change, for example).

Package **com.ibm.mqe.adapter**

This class is a descendant of **MQeTcpipLengthAdapte**. All usual adapter operations that are not listed here use the implementation provided by this super-class.

Method summary

Method	Purpose
activate	Initializes a new MQeTcpipHistoryAdapter object.
close	Closes an MQeTcpipHistoryAdapter object.

MQeTcpipHistoryAdapter activate

Syntax

```
public void activate( String fileDesc,
                    Object param,
                    Object options,
                    int lrecl,
                    int noRec ) throws Exception
```

Description

Initializes a new MQeTcpipHistoryAdapter object

Parameters

String fileDesc

The String that represents this adapter ("Network:1.2.3.4:8081" for example).

Object param

The parameters to be given to the created adapter when an ACCEPT is issued.

Object options

A String that specifies the open options:

MQe.MQe_Adapter_LISTEN

Creates a listening socket

MQe.MQe_Adapter_ACCEPT

Accepts a connection

MQe.MQe_Adapter_LOCALHOST. Returns the local host address

MQe.MQe_Adapter_NETWORK

Returns network type (TCPIP)

MQeTcpipHistoryAdapter

MQe.MQe_Adapter_PERSIST

Copies the persistent object on the accept call

MQe.MQe_Adapter_QOSINPUTS

Returns valid Quality of Service (QOS) names

MQe.MQe_Adapter_SETSOCKET

Reserved

MQeTcpipHistoryAdapter.MQe_Adapter_HISTORY

Use history functionality

For example, to configure a history adapter to use persistence and history improvements specify MQe.MQe_Adapter_PERSIST + MQeTcpipHistoryAdapter.MQe_Adapter_HISTORY here. This is equivalent to the string "<PERSIST><HISTORY>".

int irecl Specifies the TCP/IP block size. If the value supplied is less than zero the default of 4096 is used.

int noRec The time-out value of the socket in milliseconds

Return Values

None

Exceptions

None

Example

```
myHistoryAdapter.activate("Network:localhost:8081", null,  
    MQe.MQe_Adapter_PERSIST +  
    MQeTcpipHistoryAdapter.MQe_Adapter_HISTORY,  
    4096, 300000);
```

MQeTcpipHistoryAdapter close

Syntax

```
public void close( Object opt ) throws Exception {
```

Description

Closes the current adapter, so that it can no longer be read from or written to.

Parameters

Object opt

MQe.MQe_Adapter_HEADER

A String that contains MQe.MQe_Adapter_FINAL if this is the last close for this adapter. This causes any history buffers to be emptied.

Return Values

None

Exceptions

None

Example

```
myHistoryAdapter.close(MQe.MQe_Adapter_FINAL);
```

Part 2. Exceptions and return codes

Chapter 10. Exceptions

This chapter contains details of the exceptions that can be thrown by MQSeries Everywhere. The exceptions are listed in numerical order, and are then listed alphabetically with detailed explanations.

Ordered numerically

- 000 – Except_UnCoded
- 001 – Except_Debug
- 002 – Except_NotSupported
- 003 – Except_Syntax
- 004 – Except_Type
- 005 – Except_Command
- 006 – Except_NotFound
- 007 – Except_Data
- 008 – Except_BadRequest
- 009 – Except_Stopped
- 010 – Except_Closed
- 011 – Except_Duplicate
- 012 – Except_NotAllowed
- 013 – Except_Rule
- 014 – Except_Timeout
- 017 – Except_InvalidHandle
- 018 – Except_AllocationFail
- 020 – Except_Chnl_Attributes
- 022 – Except_Chnl_Destination
- 023 – Except_Chnl_Limit
- 024 – Except_Chnl_ID
- 025 – Except_Chnl_Overrun
- 040 – Except_Trnsport_QMgr
- 041 – Except_Trnsport_Request
- 100 – Except_QMgr_NotActive
- 101 – Except_QMgr_InvalidQMgrName
- 102 – Except_QMgr_Activated
- 103 – Except_QMgr_AlreadyExists
- 104 – Except_QMgr_InvalidQName
- 105 – Except_QMgr_QExists
- 106 – Except_QMgr_UknownQMgr
- 107 – Except_QMgr_QNotEmpty
- 108 – Except_QMgr_QDoesNotExist
- 109 – Except_QMgr_QInUse
- 110 – Except_QMgr_WrongQtype
- 111 – Except_QMgr_InvalidChannel

- | • 112 – Except_QMgr_SecureMsgDecodeFailed
- | • 113 – Except_QMgr_NotConfigured
- | • 114 – Except_QMgr_Busy
- | • 120 – Except_Q_NoMsg_Available
- | • 121 – Except_Q_NoMatchingMsg
- | • 124 – Except_Q_InvalidPriority
- | • 125 – Except_Q_Full
- | • 126 – Except_Q_MsgTooLarge
- | • 127 – Except_Q_NotActive
- | • 128 – Except_Q_Active
- | • 129 – Except_Q_InvalidName
- | • 130 – Except_Q_TargetRegistryRequired
- | • 180 – Except_Uncontactable_DontTransmit
- | • 201 – Except_RasDialFailed
- | • 202 – Except_RasGetProjectionInfoFailed
- | • 203 – Except_RasHangupFailed
- | • 210 – Except_Connect_AdapterNotActive
- | • 211 – Except_Connect_InvalidDefinition
- | • 230 – Except_Con_AlreadyExists
- | • 231 – Except_Con_AliasAlreadyExists
- | • 232 – Except_Con_AdapterRequired
- | • 233 – Except_Con_InvalidName
- | • 301 – Except_Reg_NullName
- | • 302 – Except_Reg_AlreadyExists
- | • 303 – Except_Reg_DoesNotExist
- | • 304 – Except_Reg_OpenFailed
- | • 305 – Except_Reg_InvalidSession
- | • 306 – Except_Reg_NotDefined
- | • 309 – Except_Reg_AddFailed
- | • 310 – Except_Reg_DeleteFailed
- | • 311 – Except_Reg_ReadFailed
- | • 312 – Except_Reg_UpdateFailed
- | • 313 – Except_Reg_ListFailed
- | • 314 – Except_Reg_SearchFailed
- | • 315 – Except_Reg_RenameFailed
- | • 316 – Except_Reg_ResetPINFailed
- | • 317 – Except_Reg_CRTKeyDecFailed
- | • 318 – Except_Reg_CRTKeySignFailed
- | • 319 – Except_Reg_DeleteRegistryFailed
- | • 320 – Except_Reg_AlreadyOpen
- | • 321 – Except_Reg_NotSecure
- | • 350 – Except_PrivateReg_BadPIN
- | • 351 – Except_PrivateReg_ActivateFailed
- | • 352 – Except_PrivateReg_NotOpen
- | • 360 – Except_MiniCertreg_BadPIN

- | • 361 – Except_MiniCertreg_ActivateFailed
- | • 362 – Except_MiniCertreg_NotOpen
- | • 370 – Except_PublicReg_ActivateFailed
- | • 371 – Except_PublicReg_InvalidRequest
- | • 400 – Except_Admin_NotAdminMsg
- | • 401 – Except_Admin_ActionNotSupported
- | • 402 – Except_Admin_InvalidField
- | • 500 – Except_Authenticate
- | • 501 – Except_S_Cipher
- | • 502 – Except_S_InvalidSignature
- | • 503 – Except_S_CertificateExpired
- | • 504 – Except_S_InvalidAttribute
- | • 505 – Except_S_MiniCertNotAvailable
- | • 506 – Except_S_RegistryNotAvailable
- | • 507 – Except_S_BadIntegrity
- | • 508 – Except_S_NoPresetKeyAvailable
- | • 509 – Except_S_MissingSection
- | • 600 – Except_Generic_MQ_Error
- | • 601 – Except_MQ_QMgr_Not_Available
- | • 602 – Except_Characteristic_Unknown
- | • 603 – Except_Characteristic_Read_Only_Violation
- | • 604 – Except_Bridge_Not_On_PreLoad_List
- | • 605 – Except_Registry_Update_Failed
- | • 606 – Except_Characteristic_Not_Found_In_Input
- | • 607 – Except_Field_Type_Mismatch
- | • 608 – Except_Invalid_Registry_Entry_Name
- | • 609 – Except_Not_Transmission_Queue
- | • 610 – Except_Bridge_Internal_Error
- | • 611 – Except_MQ_QMgr_InvalidName
- | • 612 – Except_MQ_Q_InvalidName
- | • 613 – Except_Lost_MQ_Q_Handle
- | • 614 – Except_Resource_Not_Found
- | • 615 – Except_Transformation_Error
- | • 616 – Except_Parent_Not_Running
- | • 617 – Except_Invalid_Run_State
- | • 618 – Except_Could_Not_Delete_Myself
- | • 619 – Except_Listener_State_Error
- | • 620 – Except_Invalid_Characteristic
- | • 621 – Except_Delete_Children_First
- | • 622 – Except_InvalidName
- | • 623 – Except_Class_Not_MQeRules
- | • 624 – Except_Adapter_Not_Found
- | • 625 – Except_Object_Not_Started
- | • 626 – Except_ShutDown_Children_First
- | • 627 – Except_Invalid_Reg_Type

- 628 – Except_Invalid_Return_Type
- 629 – Except_Message_Undelivered
- 630 – Except_Bridges_Already_Loaded
- 631 – Except_No_Transformer_Found
- 632 – Except_Lost_MQ_Connection
- 633 – Except_Client_Con_Not_Available
- 634 – Except_Exit_Class_Init_Error

Ordered alphabetically

- Except_Adapter_Not_Found – 624
- Except_Admin_ActionNotSupported – 401
- Except_Admin_InvalidField – 402
- Except_Admin_NotAdminMsg – 400
- Except_AllocationFail – 018
- Except_Authenticate – 500
- Except_BadRequest – 008
- Except_Bridge_Internal_Error – 610
- Except_Bridge_Not_On_PreLoad_List – 604
- Except_Bridges_Already_Loaded – 630
- Except_Characteristic_Not_Found_In_Input – 606
- Except_Characteristic_Read_Only_Violation – 603
- Except_Characteristic_Unknown – 602
- Except_Chnl_Attributes – 020
- Except_Chnl_Destination – 022
- Except_Chnl_ID – 024
- Except_Chnl_Limit – 023
- Except_Chnl_Overrun – 025
- Except_Class_Not_MQeRules – 623
- Except_Client_Con_Not_Available – 633
- Except_Closed – 010
- Except_Command – 005
- Except_Con_AdapterRequired – 232
- Except_Con_AliasAlreadyExists – 231
- Except_Con_AlreadyExists – 230
- Except_Con_InvalidName – 233
- Except_Connect_AdapterNotActive – 210
- Except_Connect_InvalidDefinition – 211
- Except_Could_Not_Delete_Myself – 618
- Except_Data – 007

| Except_Debug – 001
| Except_Delete_Children_First – 621
| Except_Duplicate – 011
| Except_Exit_Class_Init_Error – 634
| Except_Field_Type_Mismatch – 607
| Except_Generic_MQ_Error – 600
| Except_Invalid_Characteristic – 620
| Except_InvalidHandle – 017
| Except_InvalidName – 622
| Except_Invalid_Registry_Entry_Name – 608
| Except_Invalid_Reg_Type – 627
| Except_Invalid_Return_Type – 628
| Except_Invalid_Run_State – 617
| Except_Listener_State_Error – 619
| Except_Lost_MQ_Connection – 632
| Except_Lost_MQ_Q_Handle – 613
| Except_Message_Undelivered – 629
| Except_MiniCertreg_ActivateFailed – 361
| Except_MiniCertreg_BadPIN – 360
| Except_MiniCertreg_NotOpen – 362
| Except_MQ_Q_InvalidName – 612
| Except_MQ_QMgr_InvalidName – 611
| Except_MQ_QMgr_Not_Available – 601
| Except_No_Transformer_Found – 631
| Except_NotAllowed – 012
| Except_NotFound – 006
| Except_NotSupported – 002
| Except_Not_Transmission_Queue – 609
| Except_Object_Not_Started – 625 .
| Except_Parent_Not_Running – 616
| Except_PrivateReg_ActivateFailed – 351
| Except_PrivateReg_BadPIN – 350
| Except_PrivateReg_NotOpen – 352
| Except_PublicReg_ActivateFailed – 370
| Except_PublicReg_InvalidRequest – 371
| Except_Q_Active – 128
| Except_Q_Full – 125
|

| Except_Q_InvalidName – 129
| Except_Q_InvalidPriority – 124
| Except_Q_MsgTooLarge – 126
| Except_Q_NoMatchingMsg – 121
| Except_Q_NoMsg_Available – 120
| Except_Q_NotActive – 127
| Except_Q_TargetRegistryRequired – 130
| Except_QMgr_Activated – 102
| Except_QMgr_AlreadyExists – 103
| Except_QMgr_InvalidQMgrName – 101
| Except_QMgr_InvalidQName – 104
| Except_QMgr_NotActive – 100
| Except_QMgr_QExists – 105
| Except_QMgr_Busy – 114
| Except_QMgr_InvalidChannel – 111
| Except_QMgr_NotConfigured – 113
| Except_QMgr_QDoesNotExist – 108
| Except_QMgr_QInUse – 109
| Except_QMgr_QNotEmpty – 107
| Except_QMgr_UknownQMgr – 106
| Except_QMgr_SecureMsgDecodeFailed – 112
| Except_QMgr_WrongQtype – 110
| Except_RasDialFailed – 201
| Except_RasGetProjectionInfoFailed – 202
| Except_RasHangupFailed – 203
| Except_Reg_AddFailed – 309
| Except_Reg_AlreadyExists – 302
| Except_Reg_AlreadyOpen – 320
| Except_Reg_CRTKeyDecFailed – 317
| Except_Reg_CRTKeySignFailed – 318
| Except_Reg_DeleteFailed – 310
| Except_Reg_DeleteRegistryFailed – 319
| Except_Reg_DoesNotExist – 303
| Except_Reg_InvalidSession – 305
| Except_Reg_ListFailed – 313
| Except_Reg_NotDefined – 306
| Except_Reg_NotSecure – 321
|

| Except_Reg_NullName – 301
| Except_Reg_OpenFailed – 304
| Except_Reg_ReadFailed – 311
| Except_Reg_RenameFailed – 315
| Except_Reg_ResetPINFailed – 316
| Except_Reg_SearchFailed – 314
| Except_Reg_UpdateFailed – 312
| Except_Registry_Update_Failed – 605
| Except_Resource_Not_Found – 614
| Except_Rule – 013
| Except_S_BadIntegrity – 507
| Except_S_Cipher – 501
| Except_S_CertificateExpired – 503
| Except_S_MiniCertNotAvailable – 505
| Except_S_InvalidAttribute – 504
| Except_S_InvalidSignature – 502
| Except_S_MissingSection – 509
| Except_S_NoPresetKeyAvailable – 508
| Except_S_RegistryNotAvailable – 506
| Except_ShutDown_Children_First – 626
| Except_Stopped – 009
| Except_Syntax – 003
| Except_Timeout – 014
| Except_Transformation_Error – 615
| Except_Trnsport_QMgr – 040
| Except_Trnsport_Request – 041
| Except_Type – 004
| Except_UnCoded – 000
| Except_Uncontactable_DontTransmit – 180
|

Part 3. Appendixes

Appendix. Notices

This information was developed for products and services offered in the U.S.A. IBM® may not offer the products, services, or features discussed in this document 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 document. The furnishing of this document 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.

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 publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication 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,

notices

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 Program License Agreement, or any equivalent agreement between us.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates 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.

IBM
MQSeries

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Windows NT is a registered trademark of Microsoft Corporation in the United States and in the United States and/or other countries other countries.

Other company, product, and service names may be trademarks or service marks of others.

Bibliography

Related publications:

- *MQSeries Everyplace for Multiplatforms Introduction*, GC34-5843-00
- *MQSeries Everyplace Programming Guide*, SC34-5845-00
- *MQSeries An Introduction to Messaging and Queuing*, GC33-0805-01
- *MQSeries for Windows NT V5R1 Quick Beginnings*, GC34-5389-00

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 44-1962-870229
 - From within the U.K., use 01962-870229
- Electronically, use the appropriate network ID:
 - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
 - 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.



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC34-5846-01

