

WebSphere MQ



Application Programming Reference

Note!

Before using this information and the product it supports, be sure to read the general information under Appendix I, "Notices", on page 675.

Fourth edition (March 2003)

| This edition applies to the following WebSphere MQ V5.3.1 product:

- | • WebSphere MQ for z/OS

It also applies to the following WebSphere MQ V5.3 products:

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

Unless otherwise stated, the information also applies to these products:

- MQSeries for AT&T GIS (NCR) UNIX V2.2.1
- | • MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for Compaq NonStop Kernel V5.1
- MQSeries for Compaq OpenVMS Alpha V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for SINIX and DC/OSx V2.2.1
- MQSeries for Sun Solaris, Intel Platform Edition, V5.1

© Copyright International Business Machines Corporation 1994, 2003. All rights reserved.

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

Contents

Tables xv

About this book xvii

Who this book is for xvii

What you need to know to understand this book xviii

Terms used in this book xviii

Language compilers and assemblers xix

How to use this book xxii

Appearance of text in this book xxii

WebSphere MQ for z/OS for the WebSphere

Application Server user xxii

Summary of changes xxv

Changes for this edition (SC34-6062-03) xxv

Changes for the previous edition (SC34-6062-02) xxv

Changes for the earlier editions (SC34-6062-00 and

-01) xxv

Part 1. Data type descriptions 1

Chapter 1. Introduction 7

Elementary data types 7

MQBYTE – Byte 8

MQBYTEn – String of n bytes 8

MQCHAR – Single-byte character 8

MQCHARn – String of n single-byte characters 9

MQHCONN – Connection handle 9

MQHOBJ – Object handle 9

MQLONG – Long integer 9

MQPID – Process Id 9

MQPTR – Pointer 9

MQTID – Thread Id 10

PMQCHAR – A pointer to data of type

MQCHAR 10

PMQLONG – A pointer to data of type

MQLONG 10

PMQMD – A pointer to structure of type MQMD 10

C declarations 10

COBOL declarations 13

PL/I declarations 13

System/390 assembler declarations 14

TAL declarations 15

Visual Basic declarations 15

Structure data types – introduction 17

Summary 17

Rules for structure data types 18

Conventions used in the descriptions 18

C programming 19

Header files 19

Functions 19

Parameters with undefined data type 20

Data types 20

Manipulating binary strings 20

Manipulating character strings 20

Initial values for structures 21

Initial values for dynamic structures 21

Use from C++ 22

Notational conventions 22

COBOL programming 22

COPY files 22

Structures 23

Pointers 24

Named constants 24

Notational conventions 25

PL/I programming 25

INCLUDE files 25

Structures 26

Named constants 26

Notational conventions 26

System/390 assembler programming 26

Macros 26

Structures 27

CMQVERA macro 29

Notational conventions 29

Visual Basic programming 29

Header files 30

Parameters of the MQI calls 30

Initial values for structures 30

Notational conventions 30

Chapter 2. MQAIR – Authentication information record. 31

Overview 31

Fields 31

AuthInfoConnName (MQCHAR264) 31

AuthInfoType (MQLONG) 32

LDAPPassword (MQCHAR32) 32

LDAPUserNameLength (MQLONG) 32

LDAPUserNameOffset (MQLONG) 32

LDAPUserNamePtr (PMQCHAR) 33

StrucId (MQCHAR4) 33

Version (MQLONG) 34

Initial values and language declarations 34

C declaration 34

COBOL declaration 35

PL/I declaration 35

Visual Basic declaration 35

Chapter 3. MQBO – Begin options 37

Overview 37

Fields 37

Options (MQLONG) 37

StrucId (MQCHAR4) 37

Version (MQLONG) 38

Initial values and language declarations 38

C declaration 38

COBOL declaration 38

PL/I declaration 39

Visual Basic declaration 39

Chapter 4. MQCIH – CICS bridge header 41

Overview	42
Fields	43
AbendCode (MQCHAR4).	43
ADSDescriptor (MQLONG)	43
AttentionId (MQCHAR4).	44
Authenticator (MQCHAR8)	44
CancelCode (MQCHAR4).	44
CodedCharSetId (MQLONG)	45
CompCode (MQLONG)	45
ConversationalTask (MQLONG)	45
CursorPosition (MQLONG)	45
Encoding (MQLONG)	45
ErrorOffset (MQLONG)	45
Facility (MQBYTE8)	46
FacilityKeepTime (MQLONG)	46
FacilityLike (MQCHAR4).	46
Flags (MQLONG)	46
Format (MQCHAR8)	47
Function (MQCHAR4).	47
GetWaitInterval (MQLONG).	48
InputItem (MQLONG).	48
LinkType (MQLONG)	48
NextTransactionId (MQCHAR4)	49
OutputDataLength (MQLONG).	49
Reason (MQLONG)	49
RemoteSysId (MQCHAR4)	49
RemoteTransId (MQCHAR4)	50
ReplyToFormat (MQCHAR8)	50
Reserved1 (MQCHAR8)	50
Reserved2 (MQCHAR8)	50
Reserved3 (MQCHAR8)	50
Reserved4 (MQLONG)	50
ReturnCode (MQLONG)	50
StartCode (MQCHAR4)	51
StrucId (MQCHAR4)	52
StrucLength (MQLONG)	52
TaskEndStatus (MQLONG)	52
TransactionId (MQCHAR4)	53
UOWControl (MQLONG)	53
Version (MQLONG)	54
Initial values and language declarations	54
C declaration	55
COBOL declaration.	56
PL/I declaration.	57
System/390 assembler declaration.	58
Visual Basic declaration	59

Chapter 5. MQCNO – Connect options 61

Overview	61
Fields	62
ClientConnOffset (MQLONG)	62
ClientConnPtr (MQPTR)	62
ConnTag (MQBYTE128)	64
Options (MQLONG)	65
SSLConfigOffset (MQLONG)	69
SSLConfigPtr (PMQSCO).	69
StrucId (MQCHAR4)	70
Version (MQLONG)	70

Initial values and language declarations	71
C declaration	71
COBOL declaration.	72
PL/I declaration.	72
System/390 assembler declaration.	72
Visual Basic declaration	72

Chapter 6. MQDH – Distribution header 75

Overview	75
Fields	76
CodedCharSetId (MQLONG)	76
Encoding (MQLONG)	77
Flags (MQLONG)	77
Format (MQCHAR8)	78
ObjectRecOffset (MQLONG).	78
PutMsgRecFields (MQLONG)	78
PutMsgRecOffset (MQLONG)	79
RecsPresent (MQLONG)	79
StrucId (MQCHAR4)	79
StrucLength (MQLONG)	79
Version (MQLONG)	80
Initial values and language declarations	80
C declaration	80
COBOL declaration.	81
PL/I declaration.	81
Visual Basic declaration	82

Chapter 7. MQDLH – Dead-letter header 83

Overview	83
Fields	85
CodedCharSetId (MQLONG)	85
DestQMgrName (MQCHAR48).	85
DestQName (MQCHAR48)	86
Encoding (MQLONG)	86
Format (MQCHAR8)	86
PutApplName (MQCHAR28)	86
PutApplType (MQLONG)	87
PutDate (MQCHAR8)	87
PutTime (MQCHAR8)	87
Reason (MQLONG)	88
StrucId (MQCHAR4)	89
Version (MQLONG)	89
Initial values and language declarations	90
C declaration	90
COBOL declaration.	91
PL/I declaration.	91
System/390 assembler declaration.	92
TAL declaration	92
Visual Basic declaration	92

Chapter 8. MQGMO – Get-message options 95

Overview	95
Fields	96
GroupStatus (MQCHAR)	96
MatchOptions (MQLONG)	96
MsgToken (MQBYTE16)	99
Options (MQLONG)	100
Reserved1 (MQCHAR)	123
ResolvedQName (MQCHAR48)	123

ReturnedLength (MQLONG)	124
Segmentation (MQCHAR)	124
SegmentStatus (MQCHAR)	125
Signal1 (MQLONG)	125
Signal2 (MQLONG)	126
StrucId (MQCHAR4)	126
Version (MQLONG)	127
WaitInterval (MQLONG)	127
Initial values and language declarations	128
C declaration	128
COBOL declaration	129
PL/I declaration	129
System/390 assembler declaration	130
TAL declaration	130
Visual Basic declaration	130

Chapter 9. MQIHL – IMS information

header	133
Overview	133
Fields	134
Authenticator (MQCHAR8)	134
CodedCharSetId (MQLONG)	134
CommitMode (MQCHAR)	134
Encoding (MQLONG)	135
Flags (MQLONG)	135
Format (MQCHAR8)	135
LTermOverride (MQCHAR8)	135
MFSMapName (MQCHAR8)	136
ReplyToFormat (MQCHAR8)	136
Reserved (MQCHAR)	136
SecurityScope (MQCHAR)	136
StrucId (MQCHAR4)	136
StrucLength (MQLONG)	137
TranInstanceId (MQBYTE16)	137
TranState (MQCHAR)	137
Version (MQLONG)	138
Initial values and language declarations	138
C declaration	139
COBOL declaration	139
PL/I declaration	139
System/390 assembler declaration	140
Visual Basic declaration	140

Chapter 10. MQMD – Message descriptor 141

Overview	141
Fields	144
AccountingToken (MQBYTE32)	144
ApplIdentityData (MQCHAR32)	146
ApplOriginData (MQCHAR4)	146
BackoutCount (MQLONG)	147
CodedCharSetId (MQLONG)	147
CorrelId (MQBYTE24)	149
Encoding (MQLONG)	150
Expiry (MQLONG)	151
Feedback (MQLONG)	153
Format (MQCHAR8)	157
GroupId (MQBYTE24)	164
MsgFlags (MQLONG)	165
MsgId (MQBYTE24)	170

MsgSeqNumber (MQLONG)	172
MsgType (MQLONG)	172
Offset (MQLONG)	173
OriginalLength (MQLONG)	174
Persistence (MQLONG)	175
Priority (MQLONG)	176
PutApplName (MQCHAR28)	177
PutApplType (MQLONG)	178
PutDate (MQCHAR8)	181
PutTime (MQCHAR8)	181
ReplyToQ (MQCHAR48)	182
ReplyToQMgr (MQCHAR48)	183
Report (MQLONG)	184
StrucId (MQCHAR4)	194
UserIdentifier (MQCHAR12)	194
Version (MQLONG)	196
Initial values and language declarations	197
C declaration	198
COBOL declaration	198
PL/I declaration	199
System/390 assembler declaration	200
TAL declaration	200
Visual Basic declaration	201

Chapter 11. MQMDE – Message descriptor extension 203

Overview	203
Fields	205
CodedCharSetId (MQLONG)	205
Encoding (MQLONG)	206
Flags (MQLONG)	206
Format (MQCHAR8)	206
GroupId (MQBYTE24)	207
MsgFlags (MQLONG)	207
MsgSeqNumber (MQLONG)	207
Offset (MQLONG)	207
OriginalLength (MQLONG)	207
StrucId (MQCHAR4)	207
StrucLength (MQLONG)	207
Version (MQLONG)	208
Initial values and language declarations	208
C declaration	208
COBOL declaration	209
PL/I declaration	209
System/390 assembler declaration	210
Visual Basic declaration	210

Chapter 12. MQOD – Object descriptor 211

Overview	211
Fields	212
AlternateSecurityId (MQBYTE40)	212
AlternateUserId (MQCHAR12)	213
DynamicQName (MQCHAR48)	214
InvalidDestCount (MQLONG)	214
KnownDestCount (MQLONG)	215
ObjectName (MQCHAR48)	215
ObjectQMgrName (MQCHAR48)	216
ObjectRecOffset (MQLONG)	217
ObjectRecPtr (MQPTR)	218
ObjectType (MQLONG)	218

RecsPresent (MQLONG)	218
ResolvedQMgrName (MQCHAR48)	219
ResolvedQName (MQCHAR48)	219
ResponseRecOffset (MQLONG)	220
ResponseRecPtr (MQPTR)	220
StrucId (MQCHAR4)	221
UnknownDestCount (MQLONG)	221
Version (MQLONG)	221
Initial values and language declarations	222
C declaration	222
COBOL declaration	223
PL/I declaration	224
System/390 assembler declaration	224
TAL declaration	225
Visual Basic declaration	225

Chapter 13. MQOR – Object record 227

Overview	227
Fields	227
ObjectName (MQCHAR48)	227
ObjectQMgrName (MQCHAR48)	227
Initial values and language declarations	228
C declaration	228
COBOL declaration	228
PL/I declaration	228
Visual Basic declaration	228

Chapter 14. MQPMO – Put-message options 229

Overview	229
Fields	230
Context (MQHOBj)	230
InvalidDestCount (MQLONG)	230
KnownDestCount (MQLONG)	230
Options (MQLONG)	231
PutMsgRecFields (MQLONG)	240
PutMsgRecOffset (MQLONG)	241
PutMsgRecPtr (MQPTR)	242
RecsPresent (MQLONG)	242
ResolvedQMgrName (MQCHAR48)	243
ResolvedQName (MQCHAR48)	243
ResponseRecOffset (MQLONG)	243
ResponseRecPtr (MQPTR)	244
StrucId (MQCHAR4)	245
Timeout (MQLONG)	245
UnknownDestCount (MQLONG)	245
Version (MQLONG)	245
Initial values and language declarations	246
C declaration	246
COBOL declaration	247
PL/I declaration	247
System/390 assembler declaration	248
TAL declaration	248
Visual Basic declaration	249

Chapter 15. MQPMR – Put-message record 251

Overview	251
Fields	252
AccountingToken (MQBYTE32)	252

CorrelId (MQBYTE24)	252
Feedback (MQLONG)	252
GroupId (MQBYTE24)	252
MsgId (MQBYTE24)	253
Initial values and language declarations	253
C declaration	253
COBOL declaration	254
PL/I declaration	254
Visual Basic declaration	254

Chapter 16. MQRFH – Rules and formatting header 255

Overview	255
Fields	255
CodedCharSetId (MQLONG)	255
Encoding (MQLONG)	256
Flags (MQLONG)	256
Format (MQCHAR8)	256
NameValueString (MQCHARn)	257
StrucId (MQCHAR4)	257
StrucLength (MQLONG)	258
Version (MQLONG)	258
Initial values and language declarations	258
C declaration	259
COBOL declaration	259
PL/I declaration	259
System/390 assembler declaration	259
Visual Basic declaration	260

Chapter 17. MQRFH2 – Rules and formatting header 2 261

Overview	261
Fields	262
CodedCharSetId (MQLONG)	262
Encoding (MQLONG)	262
Flags (MQLONG)	263
Format (MQCHAR8)	263
NameValueCCSID (MQLONG)	263
NameValueData (MQCHARn)	263
NameValueLength (MQLONG)	266
StrucId (MQCHAR4)	266
StrucLength (MQLONG)	266
Version (MQLONG)	267
Initial values and language declarations	267
C declaration	267
COBOL declaration	268
PL/I declaration	268
System/390 assembler declaration	268
Visual Basic declaration	269

Chapter 18. MQRMH – Reference message header 271

Overview	271
Fields	272
CodedCharSetId (MQLONG)	273
DataLogicalLength (MQLONG)	273
DataLogicalOffset (MQLONG)	273
DataLogicalOffset2 (MQLONG)	274
DestEnvLength (MQLONG)	274
DestEnvOffset (MQLONG)	274

DestNameLength (MQLONG)	274
DestNameOffset (MQLONG)	275
Encoding (MQLONG)	275
Flags (MQLONG)	275
Format (MQCHAR8)	276
ObjectInstanceId (MQBYTE24)	276
ObjectType (MQCHAR8)	276
SrcEnvLength (MQLONG)	276
SrcEnvOffset (MQLONG)	276
SrcNameLength (MQLONG)	277
SrcNameOffset (MQLONG)	277
StrucId (MQCHAR4)	277
StrucLength (MQLONG)	278
Version (MQLONG)	278
Initial values and language declarations	278
C declaration	279
COBOL declaration	279
PL/I declaration	280
System/390 assembler declaration	280
Visual Basic declaration	281

Chapter 19. MQRR – Response record 283

Overview.	283
Fields	283
CompCode (MQLONG)	283
Reason (MQLONG)	283
Initial values and language declarations	284
C declaration	284
COBOL declaration	284
PL/I declaration	284
Visual Basic declaration	284

Chapter 20. MQSCO – SSL configuration options 285

Overview.	285
Fields	285
AuthInfoRecCount (MQLONG)	285
AuthInfoRecOffset (MQLONG)	286
AuthInfoRecPtr (PMQAIR)	286
CryptoHardware (MQCHAR256)	286
KeyRepository (MQCHAR256)	287
StrucId (MQCHAR4)	288
Version (MQLONG)	288
Initial values and language declarations	288
C declaration	289
COBOL declaration	289
PL/I declaration	289
Visual Basic declaration	290

Chapter 21. MQTM – Trigger message 291

Overview.	291
Fields	293
AppId (MQCHAR256)	293
AppType (MQLONG)	293
EnvData (MQCHAR128)	294
ProcessName (MQCHAR48)	294
QName (MQCHAR48)	295
StrucId (MQCHAR4)	295
TriggerData (MQCHAR64)	295
UserData (MQCHAR128)	296

Version (MQLONG)	296
Initial values and language declarations	296
C declaration	297
COBOL declaration	297
PL/I declaration	297
System/390 assembler declaration	298
TAL declaration	298
Visual Basic declaration	298

Chapter 22. MQTMC2 – Trigger message 2 (character format) 299

Overview.	299
Fields	300
AppId (MQCHAR256)	300
AppType (MQCHAR4)	300
EnvData (MQCHAR128)	300
ProcessName (MQCHAR48)	300
QMgrName (MQCHAR48)	300
QName (MQCHAR48)	300
StrucId (MQCHAR4)	300
TriggerData (MQCHAR64)	300
UserData (MQCHAR128)	301
Version (MQCHAR4)	301
Initial values and language declarations	301
C declaration	301
COBOL declaration	302
PL/I declaration	302
System/390 assembler declaration	302
TAL declaration	303
Visual Basic declaration	303

Chapter 23. MQWIH – Work information header 305

Overview.	305
Fields	305
CodedCharSetId (MQLONG)	305
Encoding (MQLONG)	306
Flags (MQLONG)	306
Format (MQCHAR8)	306
MsgToken (MQBYTE16)	307
Reserved (MQCHAR32)	307
ServiceName (MQCHAR32)	307
ServiceStep (MQCHAR8)	307
StrucId (MQCHAR4)	307
StrucLength (MQLONG)	307
Version (MQLONG)	308
Initial values and language declarations	308
C declaration	308
COBOL declaration	309
PL/I declaration	309
System/390 assembler declaration	309
Visual Basic declaration	310

Chapter 24. MQXP – Exit parameter block 311

Overview.	311
Fields	311
ExitCommand (MQLONG)	311
ExitId (MQLONG)	312
ExitParmCount (MQLONG)	312

ExitReason (MQLONG)	312
ExitResponse (MQLONG)	313
ExitUserArea (MQBYTE16)	313
Reserved (MQLONG)	313
StrucId (MQCHAR4)	314
Version (MQLONG)	314
Language declarations	314
C declaration	314
COBOL declaration	314
PL/I declaration	315
System/390 assembler declaration	315

Chapter 25. MQXQH – Transmission-queue header 317

Overview	317
Fields	320
MsgDesc (MQMD1)	320
RemoteQMgrName (MQCHAR48)	320
RemoteQName (MQCHAR48)	321
StrucId (MQCHAR4)	321
Version (MQLONG)	321
Initial values and language declarations	321
C declaration	322
COBOL declaration	322
PL/I declaration	323
System/390 assembler declaration	324
TAL declaration	324
Visual Basic declaration	324

Part 2. Function calls 327

Chapter 26. Call descriptions 331

Conventions used in the call descriptions	331
Using the calls in the C language	333
Declaring the Buffer parameter	333

Chapter 27. MQBACK – Back out changes 335

Syntax	335
Parameters	335
Hconn (MQHCONN) – input	335
CompCode (MQLONG) – output	335
Reason (MQLONG) – output	335
Usage notes	336
Language invocations	338
C invocation	338
COBOL invocation	339
PL/I invocation	339
System/390 assembler invocation	339
TAL invocation	339
Visual Basic invocation	339

Chapter 28. MQBEGIN – Begin unit of work 341

Syntax	341
Parameters	341
Hconn (MQHCONN) – input	341
BeginOptions (MQBO) – input/output	341
CompCode (MQLONG) – output	341

Reason (MQLONG) – output	342
Usage notes	342
Language invocations	344
C invocation	344
COBOL invocation	344
PL/I invocation	344
Visual Basic invocation	344

Chapter 29. MQCLOSE – Close object 345

Syntax	345
Parameters	345
Hconn (MQHCONN) – input	345
Hobj (MQHOBJ) – input/output	345
Options (MQLONG) – input	345
CompCode (MQLONG) – output	347
Reason (MQLONG) – output	347
Usage notes	348
Language invocations	350
C invocation	350
COBOL invocation	351
PL/I invocation	351
System/390 assembler invocation	351
TAL invocation	351
Visual Basic invocation	351

Chapter 30. MQCMIT – Commit changes 353

Syntax	353
Parameters	353
Hconn (MQHCONN) – input	353
CompCode (MQLONG) – output	353
Reason (MQLONG) – output	354
Usage notes	354
Language invocations	356
C invocation	356
COBOL invocation	357
PL/I invocation	357
System/390 assembler invocation	357
TAL invocation	357
Visual Basic invocation	357

Chapter 31. MQCONN – Connect queue manager 359

Syntax	359
Parameters	359
QMgrName (MQCHAR48) – input	359
Hconn (MQHCONN) – output	361
CompCode (MQLONG) – output	362
Reason (MQLONG) – output	363
Usage notes	364
Language invocations	366
C invocation	366
COBOL invocation	366
PL/I invocation	366
System/390 assembler invocation	367
TAL invocation	367
Visual Basic invocation	367

Chapter 32. MQCONNX – Connect queue manager (extended) 369

Syntax	369
Parameters	369
QMgrName (MQCHAR48) – input	369
ConnectOpts (MQCNO) – input/output	369
Hconn (MQHCONN) – output	369
CompCode (MQLONG) – output	369
Reason (MQLONG) – output	369
Usage notes	371
Language invocations	371
C invocation	371
COBOL invocation	371
PL/I invocation	371
System/390 assembler invocation	372
Visual Basic invocation	372

Chapter 33. MQDISC – Disconnect queue manager 373

Syntax	373
Parameters	373
Hconn (MQHCONN) – input/output	373
CompCode (MQLONG) – output	373
Reason (MQLONG) – output	374
Usage notes	375
Language invocations	376
C invocation	376
COBOL invocation	376
PL/I invocation	376
System/390 assembler invocation	376
TAL invocation	376
Visual Basic invocation	377

Chapter 34. MQGET – Get message 379

Syntax	379
Parameters	379
Hconn (MQHCONN) – input	379
Hobj (MQHOBJ) – input	379
MsgDesc (MQMD) – input/output	379
GetMsgOpts (MQGMO) – input/output	380
BufferLength (MQLONG) – input	380
Buffer (MQBYTE×BufferLength) – output	380
DataLength (MQLONG) – output	381
CompCode (MQLONG) – output	381
Reason (MQLONG) – output	381
Usage notes	385
Language invocations	389
C invocation	389
COBOL invocation	389
PL/I invocation	390
System/390 assembler invocation	390
TAL invocation	390
Visual Basic invocation	391

Chapter 35. MQINQ – Inquire object attributes 393

Syntax	393
Parameters	393
Hconn (MQHCONN) – input	393

Hobj (MQHOBJ) – input	393
SelectorCount (MQLONG) – input	393
Selectors (MQLONG×SelectorCount) – input	394
IntAttrCount (MQLONG) – input	397
IntAttrs (MQLONG×IntAttrCount) – output	397
CharAttrLength (MQLONG) – input	398
CharAttrs (MQCHAR×CharAttrLength) – output	398
CompCode (MQLONG) – output	398
Reason (MQLONG) – output	398
Usage notes	400
Language invocations	401
C invocation	401
COBOL invocation	401
PL/I invocation	402
System/390 assembler invocation	402
TAL invocation	402
Visual Basic invocation	403

Chapter 36. MQOPEN – Open object 405

Syntax	405
Parameters	405
Hconn (MQHCONN) – input	405
ObjDesc (MQOD) – input/output	405
Options (MQLONG) – input	406
Hobj (MQHOBJ) – output	412
CompCode (MQLONG) – output	412
Reason (MQLONG) – output	412
Usage notes	415
Language invocations	421
C invocation	421
COBOL invocation	421
PL/I invocation	421
System/390 assembler invocation	421
TAL invocation	422
Visual Basic invocation	422

Chapter 37. MQPUT – Put message 423

Syntax	423
Parameters	423
Hconn (MQHCONN) – input	423
Hobj (MQHOBJ) – input	423
MsgDesc (MQMD) – input/output	423
PutMsgOpts (MQPMO) – input/output	424
BufferLength (MQLONG) – input	424
Buffer (MQBYTE×BufferLength) – input	425
CompCode (MQLONG) – output	425
Reason (MQLONG) – output	425
Usage notes	430
Language invocations	434
C invocation	434
COBOL invocation	434
PL/I invocation	435
System/390 assembler invocation	435
TAL invocation	435
Visual Basic invocation	435

Chapter 38. MQPUT1 – Put one message 437

Syntax	437
------------------	-----

Parameters	437
Hconn (MQHCONN) – input	437
ObjDesc (MQOD) – input/output	437
MsgDesc (MQMD) – input/output	437
PutMsgOpts (MQPMO) – input/output	438
BufferLength (MQLONG) – input	438
Buffer (MQBYTE×BufferLength) – input	438
CompCode (MQLONG) – output.	438
Reason (MQLONG) – output	438
Usage notes	443
Language invocations	445
C invocation.	445
COBOL invocation	445
PL/I invocation	445
System/390 assembler invocation.	446
TAL invocation.	446
Visual Basic invocation	446

Chapter 39. MQSET – Set object attributes 447

Syntax.	447
Parameters	447
Hconn (MQHCONN) – input	447
Hobj (MQHOBJ) – input.	447
SelectorCount (MQLONG) – input	447
Selectors (MQLONG×SelectorCount) – input	447
IntAttrCount (MQLONG) – input	448
IntAttrs (MQLONG×IntAttrCount) – input	448
CharAttrLength (MQLONG) – input	449
CharAttrs (MQCHAR×CharAttrLength) – input	449
CompCode (MQLONG) – output.	449
Reason (MQLONG) – output	449
Usage notes	451
Language invocations	452
C invocation.	452
COBOL invocation	452
PL/I invocation	452
System/390 assembler invocation.	453
TAL invocation.	453
Visual Basic invocation	453

Part 3. Attributes of objects 455

Chapter 40. Attributes for queues. . . 457

Attribute descriptions	460
AlterationDate (MQCHAR12)	460
AlterationTime (MQCHAR8)	461
BackoutRequeueQName (MQCHAR48).	461
BackoutThreshold (MQLONG)	461
BaseQName (MQCHAR48)	462
CFStrucName (MQCHAR12)	462
ClusterName (MQCHAR48)	462
ClusterNamelist (MQCHAR48)	463
CreationDate (MQCHAR12)	463
CreationTime (MQCHAR8)	464
CurrentQDepth (MQLONG)	464
DefBind (MQLONG)	464
DefinitionType (MQLONG).	465
DefInputOpenOption (MQLONG)	466
DefPersistence (MQLONG).	467

DefPriority (MQLONG)	468
DistLists (MQLONG).	468
HardenGetBackout (MQLONG)	469
IndexType (MQLONG)	470
InhibitGet (MQLONG)	472
InhibitPut (MQLONG)	473
InitiationQName (MQCHAR48)	473
MaxMsgLength (MQLONG)	474
MaxQDepth (MQLONG)	475
MsgDeliverySequence (MQLONG)	475
OpenInputCount (MQLONG)	476
OpenOutputCount (MQLONG)	477
ProcessName (MQCHAR48)	477
QDepthHighEvent (MQLONG)	477
QDepthHighLimit (MQLONG)	478
QDepthLowEvent (MQLONG)	478
QDepthLowLimit (MQLONG).	479
QDepthMaxEvent (MQLONG)	479
QDesc (MQCHAR64).	480
QName (MQCHAR48)	480
QServiceInterval (MQLONG)	480
QServiceIntervalEvent (MQLONG)	481
QSGDisp (MQLONG)	481
QType (MQLONG)	482
RemoteQMgrName (MQCHAR48)	483
RemoteQName (MQCHAR48).	483
RetentionInterval (MQLONG)	484
Scope (MQLONG).	484
Shareability (MQLONG).	485
StorageClass (MQCHAR8)	485
TriggerControl (MQLONG).	486
TriggerData (MQCHAR64)	486
TriggerDepth (MQLONG)	487
TriggerMsgPriority (MQLONG)	487
TriggerType (MQLONG).	487
Usage (MQLONG)	488
XmitQName (MQCHAR48).	488

Chapter 41. Attributes for namelists 491

Attribute descriptions	491
AlterationDate (MQCHAR12)	491
AlterationTime (MQCHAR8)	491
NameCount (MQLONG)	492
NamelistDesc (MQCHAR64)	492
NamelistName (MQCHAR48)	492
NamelistType (MQLONG)	493
Names (MQCHAR48×NameCount)	493
QSGDisp (MQLONG)	493

Chapter 42. Attributes for process definitions. 495

Attribute descriptions	495
AlterationDate (MQCHAR12)	495
AlterationTime (MQCHAR8)	495
AppId (MQCHAR256)	496
AppType (MQLONG)	496
EnvData (MQCHAR128).	497
ProcessDesc (MQCHAR64)	498
ProcessName (MQCHAR48)	498
QSGDisp (MQLONG)	498

UserData (MQCHAR128) 499

Chapter 43. Attributes for the queue manager 501

Attribute descriptions 502
AlterationDate (MQCHAR12) 502
AlterationTime (MQCHAR8) 503
AuthorityEvent (MQLONG) 503
ChannelAutoDef (MQLONG) 503
ChannelAutoDefEvent (MQLONG) 504
ChannelAutoDefExit (MQCHARn) 504
ClusterWorkloadData (MQCHAR32) 505
ClusterWorkloadExit (MQCHARn) 505
ClusterWorkloadLength (MQLONG) 505
CodedCharSetId (MQLONG) 506
CommandInputQName (MQCHAR48) 506
CommandLevel (MQLONG) 506
DeadLetterQName (MQCHAR48) 508
DefXmitQName (MQCHAR48) 510
DistLists (MQLONG) 510
ExpiryInterval (MQLONG) 510
IGQPutAuthority (MQLONG) 510
IGQUserId (MQLONG) 512
InhibitEvent (MQLONG) 512
IntraGroupQueuing (MQLONG) 512
LocalEvent (MQLONG) 513
MaxHandles (MQLONG) 513
MaxMsgLength (MQLONG) 514
MaxPriority (MQLONG) 514
MaxUncommittedMsgs (MQLONG) 514
PerformanceEvent (MQLONG) 515
Platform (MQLONG) 516
QMgrDesc (MQCHAR64) 516
QMgrIdentifier (MQCHAR48) 517
QMgrName (MQCHAR48) 517
QSGName (MQCHAR4) 517
RemoteEvent (MQLONG) 517
RepositoryName (MQCHAR48) 518
RepositoryNamelist (MQCHAR48) 518
StartStopEvent (MQLONG) 519
SyncPoint (MQLONG) 519
TriggerInterval (MQLONG) 519

Chapter 44. Attributes for authentication information objects 521

Attribute descriptions 521
AlterationDate (MQCHAR12) 521
AlterationTime (MQCHAR8) 521
AuthInfoConnName (MQCHAR264) 522
AuthInfoDesc (MQCHAR64) 522
AuthInfoName (MQCHAR48) 522
AuthInfoType (MQLONG) 522
LDAPPassword (MQCHAR32) 522
LDAPUserName
(MQ_DISTINGUISHED_NAME_LENGTH) 522

Part 4. Appendixes 525

Appendix A. Return codes 527

Completion codes 527
Reason codes 528

Appendix B. MQ constants 529

List of constants 529
MQ_* (Lengths of character string and byte fields) 529
MQACT_* (Accounting token) 530
MQACTT_* (Accounting token type) 531
MQAIR_* (Authentication information record structure identifier) 531
MQAIR_* (Authentication information record version) 531
MQAIT_* (Authentication information type) 531
MQAT_* (Application type) 531
MQBND_* (Binding) 532
MQBO_* (Begin options) 532
MQBO_* (Begin options structure identifier) 532
MQBO_* (Begin options version) 533
MQCA_* (Character attribute selector) 533
MQCADSD_* (CICS header ADS descriptor) 534
MQCC_* (Completion code) 534
MQCCSI_* (Coded character set identifier) 534
MQCCT_* (CICS header conversational task) 534
MQCFAC_* (CICS header facility) 535
MQCFUNC_* (CICS header function name) 535
MQCGWI_* (CICS header get-wait interval) 535
MQCHAD_* (Channel auto-definition) 535
MQCL_* (Correlation identifier) 536
MQCIH_* (CICS header flags) 536
MQCIH_* (CICS header length) 536
MQCIH_* (CICS header structure identifier) 536
MQCIH_* (CICS header version) 537
MQCLT_* (CICS header link type) 537
MQCMDL_* (Command level) 537
MQCNO_* (Connect options) 537
MQCNO_* (Connect options structure identifier) 538
MQCNO_* (Connect options version) 538
MQCO_* (Close options) 538
MQCODL_* (CICS header output data length) 538
MQCRC_* (CICS header return code) 538
MQCSC_* (CICS header transaction start code) 539
MQCT_* (Connection tag) 539
MQCTES_* (CICS header task end status) 539
MQCUOWC_* (CICS header unit-of-work control) 539
MQDCC_* (Convert-characters masks and factors) 540
MQDCC_* (Convert-characters options) 540
MQDH_* (Distribution header structure identifier) 540
MQDH_* (Distribution header version) 540
MQDHF_* (Distribution header flags) 541
MQDL_* (Distribution list support) 541
MQDLH_* (Dead-letter header structure identifier) 541
MQDLH_* (Dead-letter header version) 541
MQDXP_* (Data-conversion-exit parameter structure identifier) 541
MQDXP_* (Data-conversion-exit parameter structure version) 542

MQEC_* (Signal event-control-block completion code)	542	MQOL_* (Original length)	555
MQEI_* (Expiry interval)	542	MQOO_* (Open options)	555
MQENC_* (Encoding)	542	MQOT_* (Object type)	556
MQENC_* (Encoding masks)	542	MQPER_* (Persistence)	556
MQENC_* (Encoding for packed-decimal integers)	543	MQPL_* (Platform)	556
MQENC_* (Encoding for floating-point numbers)	543	MQPMO_* (Put message options)	556
MQENC_* (Encoding for binary integers)	543	MQPMO_* (Put message options structure length)	557
MQEVR_* (Event reporting)	543	MQPMO_* (Put message options structure identifier)	557
MQEXPI_* (Expiry scan interval)	543	MQPMO_* (Put message options version)	557
MQFB_* (Feedback)	544	MQPMRF_* (Put message record field flags)	557
MQFMT_* (Format)	545	MQPRI_* (Priority)	557
MQGI_* (Group identifier)	545	MQQA_* (Inhibit get)	558
MQGMO_* (Get message options)	546	MQQA_* (Inhibit put)	558
MQGMO_* (Get message options structure identifier)	546	MQQA_* (Backout hardening)	558
MQGMO_* (Get message options version)	546	MQQA_* (Queue shareability)	558
MQGS_* (Group status)	547	MQQDT_* (Queue definition type)	558
MQHC_* (Connection handle)	547	MQQSGD_* (Queue-sharing group disposition)	558
MQHO_* (Object handle)	547	MQQSIE_* (Service interval events)	559
MQIA_* (Integer attribute selector)	547	MQQT_* (Queue type)	559
MQIAUT_* (IMS authenticator)	548	MQRC_* (Reason code)	559
MQIAV_* (Integer attribute value)	549	MQRFH_* (Rules and formatting header flags)	559
MQICM_* (IMS commit mode)	549	MQRFH_* (Rules and formatting header length)	559
MQIGQ_* (Intra-group queuing)	549	MQRFH_* (Rules and formatting header structure identifier)	559
MQIGQPA_* (Intra-group queuing put authority)	549	MQRFH_* (Rules and formatting header version)	560
MQIIH_* (IMS header flags)	549	MQRL_* (Returned length)	560
MQIIH_* (IMS header length)	550	MQRMH_* (Reference message header structure identifier)	560
MQIIH_* (IMS header structure identifier)	550	MQRMH_* (Reference message header version)	560
MQIIH_* (IMS header version)	550	MQRMHF_* (Reference message header flags)	560
MQISS_* (IMS security scope)	550	MQRO_* (Report options)	561
MQIT_* (Index type)	550	MQRO_* (Report-options masks)	561
MQITII_* (IMS transaction instance identifier)	551	MQSCO_* (Queue scope)	561
MQITS_* (IMS transaction state)	551	MQSCO_* (SSL configuration options structure identifier)	561
MQMD_* (Message descriptor structure identifier)	551	MQSCO_* (SSL configuration options version)	562
MQMD_* (Message descriptor version)	551	MQSEG_* (Segmentation)	562
MQMDE_* (Message descriptor extension length)	551	MQSID_* (Security identifier)	562
MQMDE_* (Message descriptor extension structure identifier)	552	MQSIDT_* (Security identifier type)	562
MQMDE_* (Message descriptor extension version)	552	MQSP_* (Syncpoint)	562
MQMDEF_* (Message descriptor extension flags)	552	MQSS_* (Segment status)	563
MQMDS_* (Message delivery sequence)	552	MQTC_* (Trigger control)	563
MQMF_* (Message flags)	552	MQTM_* (Trigger message structure identifier)	563
MQMF_* (Message-flags masks)	553	MQTM_* (Trigger message structure version)	563
MQMI_* (Message identifier)	553	MQTMC_* (Trigger message character format structure identifier)	563
MQMO_* (Match options)	553	MQTMC_* (Trigger message character format structure version)	564
MQMT_* (Message type)	553	MQTT_* (Trigger type)	564
MQMTOK_* (Message token)	554	MQUS_* (Usage)	564
MQNC_* (Name count)	554	MQWI_* (Wait interval)	564
MQNT_* (Namelist type)	554	MQWIH_* (Workload information header flags)	564
MQOD_* (Object descriptor length)	554	MQWIH_* (Workload information header structure length)	565
MQOD_* (Object descriptor structure identifier)	554	MQWIH_* (Workload information header structure identifier)	565
MQOD_* (Object descriptor version)	554	MQWIH_* (Workload information header version)	565
MQOII_* (Object instance identifier)	555		

MQXC_* (Exit command identifier)	565
MQXCC_* (Exit response)	565
MQXDR_* (Data-conversion-exit response)	566
MQXP_* (Exit parameter block structure identifier).	566
MQXP_* (Exit parameter block version)	566
MQXQH_* (Transmission queue header structure identifier)	566
MQXQH_* (Transmission queue header version)	566
MQXR_* (Exit reason)	567
MQXT_* (Exit identifier).	567
MQXUA_* (Exit user area)	567

Appendix C. Rules for validating MQI options 569

MQOPEN call	569
MQPUT call	569
MQPUT1 call	570
MQGET call	570
MQCLOSE call	570

Appendix D. Machine encodings 571

Binary-integer encoding	571
Packed-decimal-integer encoding	572
Floating-point encoding	572
Constructing encodings	573
Analyzing encodings	573
Using bit operations	573
Using arithmetic	574
Summary of machine architecture encodings	574

Appendix E. Report options and message flags 575

Structure of the report field.	575
Analyzing the report field	576
Using bit operations	577
Using arithmetic	577
Structure of the message-flags field	578

Appendix F. Data conversion. 581

Conversion processing	581
Processing conventions	583
Conversion of report messages	587
MQDXP – Data-conversion exit parameter.	589
Overview.	589
Fields	590
C declaration	594
COBOL declaration (OS/400 only)	594
System/390 assembler declaration	595
MQXCNVC – Convert characters.	595
Syntax.	595
Parameters	595
C invocation.	600
COBOL invocation (OS/400 only)	601
System/390 assembler invocation.	601
MQ_DATA_CONV_EXIT – Data conversion exit	602
Syntax.	602
Parameters	602
Usage notes	603
C invocation.	606

COBOL invocation (OS/400 only)	606
System/390 assembler invocation.	606

Appendix G. Signal notification IPC message (Compaq NonStop Kernel only) 609

Appendix H. Code page conversion 611

Codeset names and CCSIDs	612
National languages	613
US English	614
German	615
Danish and Norwegian	616
Finnish and Swedish	617
Italian	618
Spanish	619
UK English /Gaelic	620
French.	621
Multilingual.	622
Portuguese	623
Icelandic	624
Eastern European languages	625
Cyrillic	626
Estonian	627
Latvian and Lithuanian	628
Ukrainian	629
Greek	630
Turkish	631
Hebrew	632
Arabic.	634
Farsi	635
Urdu	636
Thai	637
Lao.	638
Vietnamese	639
Japanese Latin SBCS	640
Japanese Katakana SBCS	642
Japanese Kanji/ Latin Mixed	644
Japanese Kanji/ Katakana Mixed	646
Korean	648
Simplified Chinese	649
Traditional Chinese	651
z/OS conversion support	652
OS/2 conversion support	669
OS/400 conversion support	670
Unicode conversion support	670
MQSeries OS/2 support for Unicode	670
WebSphere MQ AIX support for Unicode	670
WebSphere MQ HP-UX support for Unicode	671
WebSphere MQ (for Windows, Solaris, and Linux) and MQSeries (for Compaq NSK and Tru64) support for Unicode.	671
OS/400 support for Unicode	672
WebSphere MQ for z/OS support for Unicode	672

Appendix I. Notices 675

Trademarks	676
----------------------	-----

Index 679

Sending your comments to IBM . . . 687

Tables

1. Short names used for supported environments	xviii	38. Initial values of fields in MQCIH	54
2. Language compilers for WebSphere MQ for AIX	xix	39. Fields in MQCNO	61
3. Language compilers for MQSeries for AT&T GIS UNIX	xix	40. Initial values of fields in MQCNO	71
4. Language compilers for MQSeries for Compaq NonStop Kernel.	xix	41. Fields in MQDH	75
5. Language compilers for MQSeries for Compaq OpenVMS Alpha	xix	42. Initial values of fields in MQDH	80
6. Language compilers for MQSeries for Compaq (DIGITAL) OpenVMS	xx	43. Fields in MQDLH	83
7. Language compilers for MQSeries for Compaq Tru64 UNIX	xx	44. Initial values of fields in MQDLH	90
8. Language compilers for WebSphere MQ for HP-UX	xx	45. Fields in MQGMO	95
9. Language compilers for WebSphere MQ for iSeries	xx	46. MQGET options relating to messages in groups and segments of logical messages	116
10. Language compilers for WebSphere MQ for Linux	xx	47. Outcome when MQGET or MQCLOSE call is not consistent with group and segment information	118
11. Language compilers for MQSeries for OS/2 Warp.	xx	48. Initial values of fields in MQGMO	128
12. Language compilers for MQSeries for SINIX and DC/OSx	xxi	49. Fields in MQIIH	133
13. Language compilers for WebSphere MQ for Solaris	xxi	50. Initial values of fields in MQIIH	138
14. Language compilers for MQSeries for VSE/ESA	xxi	51. Fields in MQMD	141
15. Language compilers for WebSphere MQ for Windows	xxi	52. Initial values of fields in MQMD	197
16. Language compilers and assemblers for WebSphere MQ for z/OS.	xxi	53. Fields in MQMDE	203
17. Language compilers for VM/ESA® clients	xxii	54. Queue-manager action when MQMDE specified on MQPUT or MQPUT1.	204
18. Language compilers for Windows 98 clients	xxii	55. Initial values of fields in MQMDE	208
19. Elementary data types in C	10	56. Fields in MQOD	211
20. Elementary data types in COBOL	13	57. Initial values of fields in MQOD	222
21. Elementary data types in PL/I	13	58. Fields in MQOR	227
22. Elementary data types in System/390 assembler	14	59. Initial values of fields in MQOR	228
23. Elementary data types in TAL	15	60. Fields in MQPMO	229
24. Elementary data types in Visual Basic.	15	61. MQPUT options relating to messages in groups and segments of logical messages	235
25. Structure data types used on MQI calls (or exit functions):	17	62. Outcome when MQPUT or MQCLOSE call is not consistent with group and segment information	236
26. Structure data types used in message data:	17	63. Initial values of fields in MQPMO	246
27. C header files	19	64. Fields in MQPMR	251
28. COBOL COPY files	23	65. Fields in MQRFH	255
29. PL/I INCLUDE files	25	66. Initial values of fields in MQRFH.	258
30. Assembler macros	27	67. Fields in MQRFH2.	261
31. Visual Basic header files	30	68. Initial values of fields in MQRFH2	267
32. Fields in MQAIR.	31	69. Fields in MQRMH	271
33. Initial values of fields in MQAIR	34	70. Initial values of fields in MQRMH	278
34. Fields in MQBO	37	71. Fields in MQRR.	283
35. Initial values of fields in MQBO.	38	72. Initial values of fields in MQRR	284
36. Fields in MQCIH.	41	73. Fields in MQSCO	285
37. Contents of error information fields in MQCIH structure	43	74. Initial values of fields in MQSCO	288
		75. Fields in MQTM	291
		76. Initial values of fields in MQTM	296
		77. Fields in MQTMC2	299
		78. Initial values of fields in MQTMC2	301
		79. Fields in MQWIH	305
		80. Initial values of fields in MQWIH.	308
		81. Fields in MQXP.	311
		82. Fields in MQXQH	317
		83. Initial values of fields in MQXQH	321
		84. Effect of MQCLOSE options on various types of object and queue	347

85. Scope of nonshared handles on various platforms	362		94. Recommended or required values of queue index type when MQGMO_LOGICAL_ORDER specified	472
86. MQINQ attribute selectors for queues	394		95. Attributes for namelists	491
87. MQINQ attribute selectors for namelists	396		96. Attributes for process definitions	495
88. MQINQ attribute selectors for process definitions	396		97. Attributes for the queue manager	501
89. MQINQ attribute selectors for the queue manager	396		98. Attributes for process definitions	521
90. Valid MQOPEN options for each queue type	411		99. Summary of encodings for machine architectures	574
91. MQSET attribute selectors for queues	448		100. Fields in MQDXP	589
92. Attributes for queues	459		101. Codeset names and CCSIDs.	612
93. Recommended or required values of queue index type when MQGMO_LOGICAL_ORDER not specified	471		102. WebSphere MQ for z/OS CCSID conversion support	652

About this book

The IBM® WebSphere® MQ set of products provides application programming services, on various platforms, that allow a new style of programming. This style enables you to code indirect program-to-program communication using *message queues*.

This book gives a full description of the WebSphere MQ programming interface, the MQI, for the following products:

| WebSphere MQ for AIX®, V5.3
| WebSphere MQ for HP-UX, V5.3
| WebSphere MQ for iSeries™, V5.3
| WebSphere MQ for Linux for Intel, V5.3
| WebSphere MQ for Linux for zSeries™, V5.3
| WebSphere MQ for Solaris, V5.3 (SPARC and Intel Platform Editions)
| WebSphere MQ for Windows®, V5.3
| WebSphere MQ for z/OS™, V5.3
| MQSeries® for AT&T GIS (NCR) UNIX® V2.2.1 ¹
| MQSeries for Compaq NonStop Kernel V5.1
| MQSeries for Compaq OpenVMS Alpha V5.1
| MQSeries for OS/2® Warp V5.1
| MQSeries for SINIX and DC/OSx V2.2.1
| MQSeries for Sun Solaris, Intel Platform Edition, V5.1

Notes:

1. This book does not apply to the WebSphere MQ for iSeries, V5.3 product using the RPG programming language.

You should use the *WebSphere MQ for iSeries V5.3 Application Programming Reference (ILE RPG)*, SC34-6071 for this programming language.

2. C++

This book does *not* describe the C++ programming language binding. For information on C++ you should see the *WebSphere MQ Using C++* book.

3. API exits

This book does *not* describe API exits. For information on API exits see the *WebSphere MQ Application Programming Guide*.

For information on how to design and write applications that use the services WebSphere MQ provides, see the *WebSphere MQ Application Programming Guide*.

Who this book is for

This book is for the designers of applications that use message queuing techniques, and for programmers who have to implement these designs.

1. This platform has become NCR UNIX SVR4 MP-RAS, R3.0

What you need to know to understand this book

To write message queuing applications using WebSphere MQ, you need to know how to write programs in one of the supported programming languages:

- C or COBOL (available on all supported platforms)
- PL/I (available on AIX, OS/2, Windows 2000 and Windows NT, and z/OS)
- System/390[®] assembler (available on z/OS only)
- TAL (available on Compaq NonStop Kernel only)
- Visual Basic V4 or V5 (Windows 2000 and NT only)

If the applications you are writing are to run within a CICS[®] system, you must also be familiar with CICS on your platform and its application programming interface.

To understand this book, you do not need to have written message queuing programs before.

Terms used in this book

All new terms that this book introduces are defined in the glossary. This book uses the following shortened names:

WebSphere MQ

The WebSphere MQ set of products

CICS The CICS, or Transaction Server, product for the specific platform on which you are working.

Not all of the capabilities described in this book are available in all environments. Those calls, structures, fields, or options that are not supported everywhere are identified as such in the explanatory text. Table 1 shows the short names used in this book for the various environments, and the products to which they refer.

Table 1. Short names used for supported environments

Short name used in this book	Full product or environment name
AIX	WebSphere MQ for AIX, V5.3
Compaq NonStop Kernel	MQSeries for Compaq NonStop Kernel, V5.1
Compaq OpenVMS	MQSeries for Compaq OpenVMS Alpha, V5.1
DOS client	MQ client applications running on PC-DOS
HP-UX	WebSphere MQ for HP-UX, V5.3
Linux	WebSphere MQ for Linux for Intel and Linux for zSeries, V5.3
OS/2	MQSeries for OS/2 Warp Version 5.1
OS/400 [®]	WebSphere MQ for iSeries, V5.3
Solaris	WebSphere MQ for Solaris, V5.3
UNIX systems	The UNIX systems supported by WebSphere MQ that are not Version 5. These are: <ul style="list-style-type: none">• MQSeries for AT&T GIS UNIX, V2.2• MQSeries for SINIX and DC/OSx V2.2• MQSeries for Compaq (DIGITAL) OpenVMS VAX V2.2.1.1
Windows	WebSphere MQ for Windows, V5.3

Table 1. Short names used for supported environments (continued)

Short name used in this book	Full product or environment name
z/OS	WebSphere MQ for z/OS, V5.3

The WebSphere MQ Windows client runs on the following Windows platforms:

- Windows 98
- Windows NT[®]
- Windows 2000
- Windows XP

Language compilers and assemblers

The following tables list the language compilers and assemblers supported.

Table 2. Language compilers for WebSphere MQ for AIX

Language	Compiler
C++	IBM VisualAge [®] C++ Professional, V5.0
C	IBM C for AIX, V5 IBM VisualAge C Professional, V5.0
COBOL	Micro Focus Server Express, V2.0.10
PL/I	IBM PL/I Set for AIX V1.1

Table 3. Language compilers for MQSeries for AT&T GIS UNIX

Language	Compiler
C++	AT&T C++ language system for AT&T GIS UNIX
C	AT&T GIS High Performance C, V1.0b

Table 4. Language compilers for MQSeries for Compaq NonStop Kernel

Language	Compiler
C	D45 or later using WIDE memory model (32-bit integers)
COBOL	D45 or later
TAL	D60 or later
Note: For more detail of the support in different environments, see the <i>WebSphere MQ Application Programming Guide</i> .	

Table 5. Language compilers for MQSeries for Compaq OpenVMS Alpha

Language	Compiler
C++	DEC C++, V5.2 (AXP)
C	DEC C, V5.0
COBOL	DEC COBOL, V2.2 (AXP)

About this book

Table 6. Language compilers for MQSeries for Compaq (DIGITAL) OpenVMS

Language	Compiler
C++	DEC C++, V5.0 (VAX)
C	DEC C, V5.0
COBOL	DEC COBOL, V5.0 (VAX)

Table 7. Language compilers for MQSeries for Compaq Tru64 UNIX

Language	Compiler
C++	Compaq C++ for Tru64 UNIX Version 6.2
C	Compaq C for Tru64 UNIX
COBOL	Micro Focus COBOL for UNIX Version 4.1B Micro Focus COBOL for UNIX Version 4.1.00G

Table 8. Language compilers for WebSphere MQ for HP-UX

Language	Compiler
C++	aC++
C	C bundled compiler C Softbench, V7.0
COBOL	Micro Focus Server Express, V2.0.10

Table 9. Language compilers for WebSphere MQ for iSeries

Language	Compiler
C++	IBM ILE C++ for AS/400® (program 5799-GDW) IBM VisualAge for C++ for AS/400 compiler (program 5769-CX4)
C	IBM ILE C V5R1M0
COBOL	IBM ILE COBOL V5R1M0
RPG	IBM ILE RPG IV V5R1M0

Table 10. Language compilers for WebSphere MQ for Linux

Language	Compiler
C++	GNU g++ v3.0
C	GNU gcc v3.0

Table 11. Language compilers for MQSeries for OS/2 Warp

Language	Compiler
C++	IBM C++ compiler, V3.6 IBM VisualAge for C++ for OS/2, V3.0
C	Borland C++, V2 (C bindings only) IBM C compiler, V3.6 IBM VisualAge for C++ for OS/2, V3.0 (C bindings only)
COBOL	IBM VisualAge for COBOL for OS/2, V1.1 Micro Focus COBOL, V4.0
PL/I	IBM PL/I for OS/2, V1.2 IBM VisualAge for PL/I for OS/2

Table 12. Language compilers for MQSeries for SINIX and DC/OSx

Language	Compiler
C	DC/OSx: C4.0 compiler, V4.0.1 SINIX: C compiler (C-DS, MIPS), V1.1
COBOL	Micro Focus COBOL, V3.2

Table 13. Language compilers for WebSphere MQ for Solaris

Language	Compiler
C++	Sun WorkShop compiler C++, V5.0, V6.0
C	Sun WorkShop compiler C, V5.0, V6.0
COBOL	Micro Focus Server Express, V2.0.10

Table 14. Language compilers for MQSeries for VSE/ESA

Language	Compiler
C	IBM C for VSE/ESA™, V1.1
COBOL	IBM COBOL for VSE/ESA, V1.1
PL/I	IBM PL/I for VSE/ESA, V1.1

Table 15. Language compilers for WebSphere MQ for Windows

Language	Compiler
Basic	Microsoft® Visual Basic for Windows, V5.0
C++	IBM C++ compiler, V3.6.4 IBM VisualAge for C++ for Windows, V3.5 IBM VisualAge for C++ Professional, V4.0 Microsoft Visual C++ V4, V5, & V6
C	IBM C compiler, V3.6.4 IBM VisualAge for C++ for Windows, V3.5 Microsoft Visual C++ V4, V5, & V6
COBOL	IBM VisualAge COBOL Enterprise, V3.0.1 IBM VisualAge COBOL for Windows NT, V2.1 Micro Focus Object COBOL for Windows NT, V3.3 or V4.0
PL/I	IBM PL/I for Windows, V1.2 IBM VisualAge for PL/I for Windows IBM VisualAge PL/I Enterprise, V2.1

Table 16. Language compilers and assemblers for WebSphere MQ for z/OS

Language	Compiler/Assembler
Assembler	Assembler H assembler IBM High Level Assembler/MVS assembler
C++	IBM OS/390® C/C++, V2R4
C	C/370, Release 2.1.0 IBM OS/390 C/C++, V2R4 IBM SAA® AD/Cycle® C/370
COBOL	IBM SAA AD/Cycle COBOL/370 VS COBOL II COBOL for OS/390 & VM

About this book

Table 16. Language compilers and assemblers for WebSphere MQ for z/OS (continued)

Language	Compiler/Assembler
PL/I	IBM SAA AD/Cycle PL/I Compiler OS PL/I Optimizing compiler

Table 17. Language compilers for VM/ESA® clients

Language	Compiler
Assembler	IBM Assembler
C	IBM C for VM Release, 3.1
COBOL	IBM VS COBOL II
PL/I	IBM OS/PL/I, Release 2.3
REXX	IBM VM/ESA REXX/VM

Table 18. Language compilers for Windows 98 clients

Language	Compiler
C++	IBM VisualAge for C++ for Windows, V3.5 Microsoft Visual C++, V4.0
C	Microsoft Visual C++, V4.0
COBOL	Micro Focus COBOL Workbench, V4.0

How to use this book

This book enables you to find out quickly, for example, how to use a particular call or how to correct a particular error situation.

The book presents detailed reference information about the WebSphere MQ programming interface, called the Message Queue Interface (MQI). It describes the:

- Data types that the MQI calls use
- Parameters and return codes for the calls
- Attributes of WebSphere MQ objects
- Values of constants you need to use when you write WebSphere MQ programs
- Reason codes that may occur when you run your programs

Appearance of text in this book

This book uses the following type styles:

MQOPEN

Example of the name of a call

CompCode

Example of the name of a parameter of a call, a field in a structure, or the attribute of an object

MQMD

Example of the name of a data type or structure

MQCC_FAILED

Example of the name of a constant

WebSphere MQ for z/OS for the WebSphere Application Server user

As explained in *WebSphere MQ for z/OS Concepts and Planning Guide*, WebSphere MQ for z/OS Version 5 Release 3.1 provides JMS support for WebSphere

| Application Server embedded messaging. Embedded messaging is implemented by
| WebSphere Application Server using either the supplied WebSphere MQ for z/OS
| Version 5 Release 3.1 reduced function queue manager, or a WebSphere MQ for
| z/OS Version 5 Release 3.1 full function queue manager.

| Throughout this book, we identify the commands that do not apply to, or are
| restricted in, the reduced function form of WebSphere MQ for z/OS supplied with
| WebSphere Application Server. The list below summarizes the functions that are
| not available in the reduced function form of WebSphere MQ:

- | • Queue sharing groups
- | • Clusters
- | • Channels other than server-connection channels
- | • TCP types other than OESOCKET
- | • LU62 communications
- | • Events
- | • CICS, IMS, or RRS connection
- | • IMS bridge

About this book

Summary of changes

This section describes changes in this edition of *WebSphere MQ Application 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 (SC34-6062-03)

Changes in this edition include:

- The addition of information for users of the reduced function form of WebSphere MQ for z/OS supplied with WebSphere Application Server, as explained in “WebSphere MQ for z/OS for the WebSphere Application Server user” on page xxii.
- Some clarifications and corrections.

Changes for the previous edition (SC34-6062-02)

This edition provides additions and clarifications for users of Version 5.1 of MQSeries for Compaq NonStop Kernel, MQSeries for Compaq OpenVMS Alpha, and MQSeries for Compaq Tru64 UNIX.

Changes for the earlier editions (SC34-6062-00 and -01)

The first two editions for WebSphere MQ included the following major changes:

- Changes were made throughout the book to reflect the rebranding of MQSeries to WebSphere MQ.
- The platforms Windows XP, Linux for zSeries, and Linux for Intel were added.
- WebSphere MQ is now fully integrated with the Secure Sockets Layer (SSL) protocol. The following data structures have been added for support with the Secure Sockets Layer (SSL):

- MQAIR – Authentication information record

The MQAIR structure allows an application running as a WebSphere MQ client to specify information about an authenticator that is to be used for the client connection. The structure is an input parameter on the MQCONN call.

- MQSCO – SSL configuration options

The MQSCO structure (in conjunction with the SSL fields in the MQCD structure) allows an application running as a WebSphere MQ client to specify configuration options that control the use of SSL for the client connection when the channel protocol is TCP/IP. The Structure is an input parameter on the MQCONN call.

For details of the SSL implementation on WebSphere MQ, see the *WebSphere MQ Security* book

- Appendix H – “Code page conversion” has been restructured.
- Addition of the MQSeries for OS/390 V2.2 product.
- Addition of the MQSeries for Compaq Tru64 UNIX, V5.1 product.
- Addition of the Rules and Formatting Header data type structures.
- The fields in the data type structures, and attributes for objects, are now listed in alphabetical order.

Changes

- The information about attributes for queues has been merged into a single section.
- The appendix on return codes has been restructured and contains the associated completion code, or codes, for each return code.

Part 1. Data type descriptions

Chapter 1. Introduction	7
Elementary data types	7
MQBYTE – Byte	8
MQBYTEn – String of n bytes	8
MQCHAR – Single-byte character	8
MQCHARn – String of n single-byte characters	9
MQHCONN – Connection handle	9
MQHOBJ – Object handle	9
MQLONG – Long integer	9
MQPID – Process Id	9
MQPTR – Pointer	9
MQTID – Thread Id	10
PMQCHAR – A pointer to data of type MQCHAR	10
PMQLONG – A pointer to data of type MQLONG	10
PMQMD – A pointer to structure of type MQMD	10
C declarations	10
COBOL declarations	13
PL/I declarations	13
System/390 assembler declarations	14
TAL declarations	15
Visual Basic declarations	15
Structure data types – introduction	17
Summary	17
Rules for structure data types	18
Conventions used in the descriptions	18
C programming	19
Header files	19
Functions	19
Parameters with undefined data type	20
Data types	20
Manipulating binary strings	20
Manipulating character strings	20
Initial values for structures	21
Initial values for dynamic structures	21
Use from C++	22
Notational conventions	22
COBOL programming	22
COPY files	22
Structures	23
Pointers	24
Named constants	24
Notational conventions	25
PL/I programming	25
INCLUDE files	25
Structures	26
Named constants	26
Notational conventions	26
System/390 assembler programming	26
Macros	26
Structures	27
Specifying the name of the structure	27
Specifying the form of the structure	28
Controlling the version of the structure	28

Declaring one structure embedded within another	28
Specifying initial values for fields	29
Controlling the listing	29
CMQVERA macro	29
Notational conventions	29
Visual Basic programming	29
Header files	30
Parameters of the MQI calls	30
Initial values for structures	30
Notational conventions	30

Chapter 2. MQAIR – Authentication information record

Overview	31
Fields	31
AuthInfoConnName (MQCHAR264)	31
AuthInfoType (MQLONG)	32
LDAPPassword (MQCHAR32)	32
LDAPUserNameLength (MQLONG)	32
LDAPUserNameOffset (MQLONG)	32
LDAPUserNamePtr (PMQCHAR)	33
StrucId (MQCHAR4)	33
Version (MQLONG)	34
Initial values and language declarations	34
C declaration	34
COBOL declaration	35
PL/I declaration	35
Visual Basic declaration	35

Chapter 3. MQBO – Begin options

Overview	37
Fields	37
Options (MQLONG)	37
StrucId (MQCHAR4)	37
Version (MQLONG)	38
Initial values and language declarations	38
C declaration	38
COBOL declaration	38
PL/I declaration	39
Visual Basic declaration	39

Chapter 4. MQCIH – CICS bridge header

Overview	42
Fields	43
AbendCode (MQCHAR4)	43
ADSDescriptor (MQLONG)	43
AttentionId (MQCHAR4)	44
Authenticator (MQCHAR8)	44
CancelCode (MQCHAR4)	44
CodedCharSetId (MQLONG)	45
CompCode (MQLONG)	45
ConversationalTask (MQLONG)	45
CursorPosition (MQLONG)	45
Encoding (MQLONG)	45

Data types

ErrorOffset (MQLONG)	45	PutMsgRecFields (MQLONG)	78
Facility (MQBYTE8)	46	PutMsgRecOffset (MQLONG)	79
FacilityKeepTime (MQLONG)	46	RecsPresent (MQLONG)	79
FacilityLike (MQCHAR4)	46	StrucId (MQCHAR4)	79
Flags (MQLONG)	46	StrucLength (MQLONG)	79
Format (MQCHAR8)	47	Version (MQLONG)	80
Function (MQCHAR4)	47	Initial values and language declarations	80
GetWaitInterval (MQLONG)	48	C declaration	80
InputItem (MQLONG)	48	COBOL declaration	81
LinkType (MQLONG)	48	PL/I declaration	81
NextTransactionId (MQCHAR4)	49	Visual Basic declaration	82
OutputDataLength (MQLONG)	49	Chapter 7. MQDLH – Dead-letter header	83
Reason (MQLONG)	49	Overview	83
RemoteSysId (MQCHAR4)	49	Fields	85
RemoteTransId (MQCHAR4)	50	CodedCharSetId (MQLONG)	85
ReplyToFormat (MQCHAR8)	50	DestQMgrName (MQCHAR48)	85
Reserved1 (MQCHAR8)	50	DestQName (MQCHAR48)	86
Reserved2 (MQCHAR8)	50	Encoding (MQLONG)	86
Reserved3 (MQCHAR8)	50	Format (MQCHAR8)	86
Reserved4 (MQLONG)	50	PutApplName (MQCHAR28)	86
ReturnCode (MQLONG)	50	PutApplType (MQLONG)	87
StartCode (MQCHAR4)	51	PutDate (MQCHAR8)	87
StrucId (MQCHAR4)	52	PutTime (MQCHAR8)	87
StrucLength (MQLONG)	52	Reason (MQLONG)	88
TaskEndStatus (MQLONG)	52	StrucId (MQCHAR4)	89
TransactionId (MQCHAR4)	53	Version (MQLONG)	89
UOWControl (MQLONG)	53	Initial values and language declarations	90
Version (MQLONG)	54	C declaration	90
Initial values and language declarations	54	COBOL declaration	91
C declaration	55	PL/I declaration	91
COBOL declaration	56	System/390 assembler declaration	92
PL/I declaration	57	TAL declaration	92
System/390 assembler declaration	58	Visual Basic declaration	92
Visual Basic declaration	59	Chapter 8. MQGMO – Get-message options	95
Chapter 5. MQCNO – Connect options	61	Overview	95
Overview	61	Fields	96
Fields	62	GroupStatus (MQCHAR)	96
ClientConnOffset (MQLONG)	62	MatchOptions (MQLONG)	96
ClientConnPtr (MQPTR)	62	MsgToken (MQBYTE16)	99
ConnTag (MQBYTE128)	64	Options (MQLONG)	100
Options (MQLONG)	65	Reserved1 (MQCHAR)	123
SSLConfigOffset (MQLONG)	69	ResolvedQName (MQCHAR48)	123
SSLConfigPtr (PMQSCO)	69	ReturnedLength (MQLONG)	124
StrucId (MQCHAR4)	70	Segmentation (MQCHAR)	124
Version (MQLONG)	70	SegmentStatus (MQCHAR)	125
Initial values and language declarations	71	Signal1 (MQLONG)	125
C declaration	71	Signal2 (MQLONG)	126
COBOL declaration	72	StrucId (MQCHAR4)	126
PL/I declaration	72	Version (MQLONG)	127
System/390 assembler declaration	72	WaitInterval (MQLONG)	127
Visual Basic declaration	72	Initial values and language declarations	128
Chapter 6. MQDH – Distribution header	75	C declaration	128
Overview	75	COBOL declaration	129
Fields	76	PL/I declaration	129
CodedCharSetId (MQLONG)	76	System/390 assembler declaration	130
Encoding (MQLONG)	77	TAL declaration	130
Flags (MQLONG)	77	Visual Basic declaration	130
Format (MQCHAR8)	78	Chapter 9. MQIIH – IMS information header	133
ObjectRecOffset (MQLONG)	78		

Overview	133
Fields	134
Authenticator (MQCHAR8)	134
CodedCharSetId (MQLONG)	134
CommitMode (MQCHAR)	134
Encoding (MQLONG)	135
Flags (MQLONG)	135
Format (MQCHAR8)	135
LTermOverride (MQCHAR8)	135
MFSSMapName (MQCHAR8)	136
ReplyToFormat (MQCHAR8)	136
Reserved (MQCHAR)	136
SecurityScope (MQCHAR)	136
StrucId (MQCHAR4)	136
StrucLength (MQLONG)	137
TranInstanceId (MQBYTE16)	137
TranState (MQCHAR)	137
Version (MQLONG)	138
Initial values and language declarations	138
C declaration	139
COBOL declaration	139
PL/I declaration	139
System/390 assembler declaration	140
Visual Basic declaration	140

Chapter 10. MQMD – Message descriptor 141

Overview	141
Fields	144
AccountingToken (MQBYTE32)	144
AppIdentityData (MQCHAR32)	146
AppOriginData (MQCHAR4)	146
BackoutCount (MQLONG)	147
CodedCharSetId (MQLONG)	147
CorrelId (MQBYTE24)	149
Encoding (MQLONG)	150
Expiry (MQLONG)	151
Expired messages on z/OS	153
Feedback (MQLONG)	153
Format (MQCHAR8)	157
GroupId (MQBYTE24)	164
MsgFlags (MQLONG)	165
MsgId (MQBYTE24)	170
MsgSeqNumber (MQLONG)	172
MsgType (MQLONG)	172
Offset (MQLONG)	173
OriginalLength (MQLONG)	174
Persistence (MQLONG)	175
Priority (MQLONG)	176
PutApplName (MQCHAR28)	177
PutApplType (MQLONG)	178
PutDate (MQCHAR8)	181
PutTime (MQCHAR8)	181
ReplyToQ (MQCHAR48)	182
ReplyToQMgr (MQCHAR48)	183
Report (MQLONG)	184
StrucId (MQCHAR4)	194
UserIdentifier (MQCHAR12)	194
Version (MQLONG)	196
Initial values and language declarations	197
C declaration	198
COBOL declaration	198

PL/I declaration	199
System/390 assembler declaration	200
TAL declaration	200
Visual Basic declaration	201

Chapter 11. MQMDE – Message descriptor extension 203

Overview	203
Fields	205
CodedCharSetId (MQLONG)	205
Encoding (MQLONG)	206
Flags (MQLONG)	206
Format (MQCHAR8)	206
GroupId (MQBYTE24)	207
MsgFlags (MQLONG)	207
MsgSeqNumber (MQLONG)	207
Offset (MQLONG)	207
OriginalLength (MQLONG)	207
StrucId (MQCHAR4)	207
StrucLength (MQLONG)	207
Version (MQLONG)	208
Initial values and language declarations	208
C declaration	208
COBOL declaration	209
PL/I declaration	209
System/390 assembler declaration	210
Visual Basic declaration	210

Chapter 12. MQOD – Object descriptor 211

Overview	211
Fields	212
AlternateSecurityId (MQBYTE40)	212
AlternateUserId (MQCHAR12)	213
DynamicQName (MQCHAR48)	214
InvalidDestCount (MQLONG)	214
KnownDestCount (MQLONG)	215
ObjectName (MQCHAR48)	215
ObjectQMgrName (MQCHAR48)	216
ObjectRecOffset (MQLONG)	217
ObjectRecPtr (MQPTR)	218
ObjectType (MQLONG)	218
RecsPresent (MQLONG)	218
ResolvedQMgrName (MQCHAR48)	219
ResolvedQName (MQCHAR48)	219
ResponseRecOffset (MQLONG)	220
ResponseRecPtr (MQPTR)	220
StrucId (MQCHAR4)	221
UnknownDestCount (MQLONG)	221
Version (MQLONG)	221
Initial values and language declarations	222
C declaration	222
COBOL declaration	223
PL/I declaration	224
System/390 assembler declaration	224
TAL declaration	225
Visual Basic declaration	225

Chapter 13. MQOR – Object record 227

Overview	227
Fields	227
ObjectName (MQCHAR48)	227

Data types

ObjectQMgrName (MQCHAR48)	227
Initial values and language declarations	228
C declaration	228
COBOL declaration	228
PL/I declaration	228
Visual Basic declaration	228

Chapter 14. MQPMO – Put-message options	229
Overview	229
Fields	230
Context (MQHOBJ)	230
InvalidDestCount (MQLONG)	230
KnownDestCount (MQLONG)	230
Options (MQLONG)	231
PutMsgRecFields (MQLONG)	240
PutMsgRecOffset (MQLONG)	241
PutMsgRecPtr (MQPTR)	242
RecsPresent (MQLONG)	242
ResolvedQMgrName (MQCHAR48)	243
ResolvedQName (MQCHAR48)	243
ResponseRecOffset (MQLONG)	243
ResponseRecPtr (MQPTR)	244
StrucId (MQCHAR4)	245
Timeout (MQLONG)	245
UnknownDestCount (MQLONG)	245
Version (MQLONG)	245
Initial values and language declarations	246
C declaration	246
COBOL declaration	247
PL/I declaration	247
System/390 assembler declaration	248
TAL declaration	248
Visual Basic declaration	249

Chapter 15. MQPMR – Put-message record	251
Overview	251
Fields	252
AccountingToken (MQBYTE32)	252
CorrelId (MQBYTE24)	252
Feedback (MQLONG)	252
GroupId (MQBYTE24)	252
MsgId (MQBYTE24)	253
Initial values and language declarations	253
C declaration	253
COBOL declaration	254
PL/I declaration	254
Visual Basic declaration	254

Chapter 16. MQRFH – Rules and formatting header	255
Overview	255
Fields	255
CodedCharSetId (MQLONG)	255
Encoding (MQLONG)	256
Flags (MQLONG)	256
Format (MQCHAR8)	256
NameValueString (MQCHARn)	257
StrucId (MQCHAR4)	257
StrucLength (MQLONG)	258
Version (MQLONG)	258
Initial values and language declarations	258

C declaration	259
COBOL declaration	259
PL/I declaration	259
System/390 assembler declaration	259
Visual Basic declaration	260

Chapter 17. MQRFH2 – Rules and formatting header 2	261
Overview	261
Fields	262
CodedCharSetId (MQLONG)	262
Encoding (MQLONG)	262
Flags (MQLONG)	263
Format (MQCHAR8)	263
NameValueCCSID (MQLONG)	263
NameValueData (MQCHARn)	263
NameValueLength (MQLONG)	266
StrucId (MQCHAR4)	266
StrucLength (MQLONG)	266
Version (MQLONG)	267
Initial values and language declarations	267
C declaration	267
COBOL declaration	268
PL/I declaration	268
System/390 assembler declaration	268
Visual Basic declaration	269

Chapter 18. MQRMH – Reference message header	271
Overview	271
Fields	272
CodedCharSetId (MQLONG)	273
DataLogicalLength (MQLONG)	273
DataLogicalOffset (MQLONG)	273
DataLogicalOffset2 (MQLONG)	274
DestEnvLength (MQLONG)	274
DestEnvOffset (MQLONG)	274
DestNameLength (MQLONG)	274
DestNameOffset (MQLONG)	275
Encoding (MQLONG)	275
Flags (MQLONG)	275
Format (MQCHAR8)	276
ObjectInstanceId (MQBYTE24)	276
ObjectType (MQCHAR8)	276
SrcEnvLength (MQLONG)	276
SrcEnvOffset (MQLONG)	276
SrcNameLength (MQLONG)	277
SrcNameOffset (MQLONG)	277
StrucId (MQCHAR4)	277
StrucLength (MQLONG)	278
Version (MQLONG)	278
Initial values and language declarations	278
C declaration	279
COBOL declaration	279
PL/I declaration	280
System/390 assembler declaration	280
Visual Basic declaration	281

Chapter 19. MQRR – Response record	283
Overview	283
Fields	283

CompCode (MQLONG)	283	COBOL declaration	302
Reason (MQLONG)	283	PL/I declaration	302
Initial values and language declarations	284	System/390 assembler declaration	302
C declaration	284	TAL declaration	303
COBOL declaration	284	Visual Basic declaration	303
PL/I declaration	284		
Visual Basic declaration	284		
Chapter 20. MQSCO – SSL configuration		Chapter 23. MQWIH – Work information header	305
options	285	Overview	305
Overview	285	Fields	305
Fields	285	CodedCharSetId (MQLONG)	305
AuthInfoRecCount (MQLONG)	285	Encoding (MQLONG)	306
AuthInfoRecOffset (MQLONG)	286	Flags (MQLONG)	306
AuthInfoRecPtr (PMQAIR)	286	Format (MQCHAR8)	306
CryptoHardware (MQCHAR256)	286	MsgToken (MQBYTE16)	307
KeyRepository (MQCHAR256)	287	Reserved (MQCHAR32)	307
StrucId (MQCHAR4)	288	ServiceName (MQCHAR32)	307
Version (MQLONG)	288	ServiceStep (MQCHAR8)	307
Initial values and language declarations	288	StrucId (MQCHAR4)	307
C declaration	289	StrucLength (MQLONG)	307
COBOL declaration	289	Version (MQLONG)	308
PL/I declaration	289	Initial values and language declarations	308
Visual Basic declaration	290	C declaration	308
		COBOL declaration	309
		PL/I declaration	309
		System/390 assembler declaration	309
		Visual Basic declaration	310
Chapter 21. MQTM – Trigger message	291	Chapter 24. MQXP – Exit parameter block	311
Overview	291	Overview	311
Fields	293	Fields	311
AppId (MQCHAR256)	293	ExitCommand (MQLONG)	311
AppType (MQLONG)	293	ExitId (MQLONG)	312
EnvData (MQCHAR128)	294	ExitParmCount (MQLONG)	312
ProcessName (MQCHAR48)	294	ExitReason (MQLONG)	312
QName (MQCHAR48)	295	ExitResponse (MQLONG)	313
StrucId (MQCHAR4)	295	ExitUserArea (MQBYTE16)	313
TriggerData (MQCHAR64)	295	Reserved (MQLONG)	313
UserData (MQCHAR128)	296	StrucId (MQCHAR4)	314
Version (MQLONG)	296	Version (MQLONG)	314
Initial values and language declarations	296	Language declarations	314
C declaration	297	C declaration	314
COBOL declaration	297	COBOL declaration	314
PL/I declaration	297	PL/I declaration	315
System/390 assembler declaration	298	System/390 assembler declaration	315
TAL declaration	298		
Visual Basic declaration	298		
Chapter 22. MQTMC2 – Trigger message 2 (character format)	299	Chapter 25. MQXQH – Transmission-queue header	317
Overview	299	Overview	317
Fields	300	Fields	320
AppId (MQCHAR256)	300	MsgDesc (MQMD1)	320
AppType (MQCHAR4)	300	RemoteQMgrName (MQCHAR48)	320
EnvData (MQCHAR128)	300	RemoteQName (MQCHAR48)	321
ProcessName (MQCHAR48)	300	StrucId (MQCHAR4)	321
QMgrName (MQCHAR48)	300	Version (MQLONG)	321
QName (MQCHAR48)	300	Initial values and language declarations	321
StrucId (MQCHAR4)	300	C declaration	322
TriggerData (MQCHAR64)	300	COBOL declaration	322
UserData (MQCHAR128)	301	PL/I declaration	323
Version (MQCHAR4)	301	System/390 assembler declaration	324
Initial values and language declarations	301	TAL declaration	324
C declaration	301	Visual Basic declaration	324

Data types

Chapter 1. Introduction

This chapter introduces the data types used in the MQI, and gives you some guidance on using them in the supported programming languages.

Elementary data types

The data types used in the MQI (or in exit functions) are either:

- Elementary data types, or
- Aggregates of elementary data types (arrays or structures)

The elementary data types are described below; the structure data types are described later in this book.

The following elementary data types are used in the MQI (or in exit functions):

- MQBYTE – Byte
- MQBYTEn – String of n bytes
- MQCHAR – Single-byte character
- MQCHARn – String of n single-byte characters
- MQHCONN – Connection handle
- MQHOBJ – Object handle
- MQLONG – Long integer
- MQPID – Process Id
- MQPTR – Pointer
- MQTID – Thread Id
- PMQACH – A pointer to a data structure of type MQACH
- PMQAIR – A pointer to a data structure of type MQAIR
- PMQAXC – A pointer to a data structure of type MQAXC
- PMQAXP – A pointer to a data structure of type MQAXP
- PMQBO – A pointer to a data structure of type MQBO
- PMQBYTE – A pointer to data of type MQBYTE
- PMQBYTEn – A pointer to a data type of MQBYTEn, where n can be 8, 16, 24, 32, 40, 128
- PMQCHARn – A pointer to a data type of MQCHARn, where n can be 4, 8, 12, 20, 28, 32, 48, 64, 128, 256, 264
- PMQCIH – A pointer to a data structure of type MQCIH
- PMQCNO – A pointer to a data structure of type MQCNO
- PMQDLH – A pointer to a data structure of type MQDLH
- PMQFUNC – A pointer to a function
- PMQGMO – A pointer to a data structure of type MQGMO
- PMQHCONFIG – A pointer to a data type of MQHCONFIG
- PMQHCONN – A pointer to a data type of MQHCONN
- PMQHOBJ – A pointer to a data type of MQHOBJ
- PMQIIH – A pointer to a data structure of type MQIIH
- PPMQLONG – A pointer to a data type of PMQLONG
- PMQMD – A pointer to structure of type MQCMD
- PMQMD1 – A pointer to a data structure of type MQMD1
- PMQMDE – A pointer to a data structure of type MQMDE
- PMQOD – A pointer to a data structure of type MQOD
- PMQPMO – A pointer to a data structure of type MQPMO
- PMQPTR – A pointer to a data type of MQPTR
- PMQRFH – A pointer to a data structure of type MQRFH
- PMQRFH2 – A pointer to a data structure of type MQRFH2

Elementary data types

- | • PMQRMH – A pointer to a data structure of type MQRMH
- | • PMQTM – A pointer to a data structure of type MQTM
- | • PMQTM2 – A pointer to a data structure of type MQTM2
- | • PMQULONG – A pointer to a data type of MQULONG
- | • PMQVOID – A pointer
- | • PMQWIH – A pointer to a data structure of type MQWIH
- | • PMQXQH – A pointer to a data structure of type MQXQH

These are described in detail below, followed by examples showing how the elementary data types are declared in the supported programming languages.

MQBYTE – Byte

The MQBYTE data type represents a single byte of data. No particular interpretation is placed on the byte—it is treated as a string of bits, and not as a binary number or character. No special alignment is required.

When MQBYTE data is sent between queue managers that use different character sets or encodings, the MQBYTE data is *not* converted in any way. The *MsgId* and *CorrelId* fields in the MQMD structure are like this.

An array of MQBYTE is sometimes used to represent an area of main storage whose nature is not known to the queue manager. For example, the area may contain application message data or a structure. The boundary alignment of this area must be compatible with the nature of the data contained within it.

In the C programming language, any data type can be used for function parameters that are shown as arrays of MQBYTE. This is because such parameters are always passed by address, and in C the function parameter is declared as a pointer-to-void.

MQBYTEN – String of n bytes

Each MQBYTEN data type represents a string of *n* bytes, where *n* can take any of the following values: 8, 16, 24, 32, 40, or 128. Each byte is described by the MQBYTE data type. No special alignment is required.

If the data in the byte string is shorter than the defined length of the string, the data must be padded with nulls to fill the string.

When the queue manager returns byte strings to the application (for example, on the MQGET call), the queue manager pads with nulls to the defined length of the string.

Named constants are available that define the lengths of byte string fields; see Appendix B, “MQ constants”, on page 529.

MQCHAR – Single-byte character

The MQCHAR data type represents a single-byte character, or one byte of a double-byte or multi-byte character. No special alignment is required.

When MQCHAR data is sent between queue managers that use different character sets or encodings, the MQCHAR data usually requires conversion in order for the data to be interpreted correctly. The queue manager does this automatically for MQCHAR data in the MQMD structure. Conversion of MQCHAR data in the application message data is controlled by the MQGMO_CONVERT option

specified on the MQGET call; see the description of this option in Chapter 8, “MQGMO – Get-message options”, on page 95 for further details.

MQCHARn – String of n single-byte characters

Each MQCHARn data type represents a string of *n* characters, where *n* can take any of the following values: 4, 8, 12, 20, 28, 32, 48, 64, 128, or 256. Each character is described by the MQCHAR data type. No special alignment is required.

If the data in the string is shorter than the defined length of the string, the data must be padded with blanks to fill the string. In some cases a null character can be used to end the string prematurely, instead of padding with blanks; the null character and characters following it are treated as blanks, up to the defined length of the string. The places where a null can be used are identified in the call and data type descriptions.

When the queue manager returns character strings to the application (for example, on the MQGET call), the queue manager always pads with blanks to the defined length of the string; the queue manager does not use the null character to delimit the string.

Named constants are available that define the lengths of character string fields; see Appendix B, “MQ constants”, on page 529.

MQHCONN – Connection handle

The MQHCONN data type represents a connection handle, that is, the connection to a particular queue manager. A connection handle must be aligned on a 4-byte boundary.

Note: Applications must test variables of this type for equality only.

MQHOBJ – Object handle

The MQHOBJ data type represents an object handle that gives access to an object. An object handle must be aligned on a 4-byte boundary.

Note: Applications must test variables of this type for equality only.

MQLONG – Long integer

The MQLONG data type is a 32-bit signed binary integer that can take any value in the range -2 147 483 648 through +2 147 483 647, unless otherwise restricted by the context. For COBOL, the valid range is limited to -999 999 999 through +999 999 999. An MQLONG must be aligned on a 4-byte boundary.

MQPID – Process Id

The MQ process identifier.

This is the same identifier used in MQ trace and FFST dumps, but may be different to the operating system thread identifier.

MQPTR – Pointer

The MQPTR data type is the address of data of any type. A pointer must be aligned on its natural boundary; this is a 16-byte boundary on OS/400, and a 4-byte boundary on other platforms.

Elementary data types

Some programming languages support typed pointers; the MQI also uses these in a few cases (for example, PMQCHAR and PMQLONG in the C programming language).

MQTID – Thread Id

The MQ thread identifier.

This is the same identifier used in MQ trace and FFST dumps, but may be different to the operating system thread identifier.

PMQCHAR – A pointer to data of type MQCHAR

A pointer to data of type MQCHAR.

PMQLONG – A pointer to data of type MQLONG

A pointer to data of type MQLONG.

PMQMD – A pointer to structure of type MQMD

A pointer to structure of type MQMD.

C declarations

Table 19. Elementary data types in C

Data type	Representation
MQBYTE	typedef unsigned char MQBYTE;
MQBYTE8	typedef MQBYTE MQBYTE8[8];
MQBYTE16	typedef MQBYTE MQBYTE16[16];
MQBYTE24	typedef MQBYTE MQBYTE24[24];
MQBYTE32	typedef MQBYTE MQBYTE32[32];
MQBYTE40	typedef MQBYTE MQBYTE40[40];
MQCHAR	typedef char MQCHAR;
MQCHAR4	typedef MQCHAR MQCHAR4[4];
MQCHAR8	typedef MQCHAR MQCHAR8[8];
MQCHAR12	typedef MQCHAR MQCHAR12[12];
MQCHAR20	typedef MQCHAR MQCHAR20[20];
MQCHAR28	typedef MQCHAR MQCHAR28[28];
MQCHAR32	typedef MQCHAR MQCHAR32[32];
MQCHAR48	typedef MQCHAR MQCHAR48[48];
MQCHAR64	typedef MQCHAR MQCHAR64[64];
MQCHAR128	typedef MQCHAR MQCHAR128[128];
MQCHAR256	typedef MQCHAR MQCHAR256[256];
MQHCONFIG	typedef void* MQPOINTER MQHCONFIG;
MQHCONN	typedef MQLONG MQHCONN;
MQHOBJ	typedef MQLONG MQHOBJ;

Table 19. Elementary data types in C (continued)

Data type	Representation
MQLONG	On iSeries: typedef long MQLONG; other platforms: if defined(MQ_64_BIT) typedef int MQLONG; else typedef long MQLONG;
MQPID	typedef MQLONG MQPID;
MQPTR	typedef void MQPOINTER MQPTR;
MQTID	typedef MQLONG MQTID;
MQULONG	On iSeries: typedef unsigned long MQULONG; other platforms: if defined(MQ_64_BIT) typedef unsigned int MQULONG; else typedef unsigned long MQULONG;
PMQBO	typedef MQBO MQPOINTER PMQBO;
PMQBYTE	typedef MQBYTE MQPOINTER PMQBYTE;
PMQBYTE8	typedef MQBYTE8[8] MQPOINTER PMQBYTE8[8];
PMQBYTE16	typedef MQBYTE16[16] MQPOINTER PMQBYTE16[16];
PMQBYTE24	typedef MQBYTE24[24] MQPOINTER PMQBYTE24[24];
PMQBYTE32	typedef MQBYTE32[32] MQPOINTER PMQBYTE32[32];
PMQBYTE40	typedef MQBYTE40[40] MQPOINTER PMQBYTE40[40];
PMQBYTE128	typedef MQBYTE128[128] MQPOINTER PMQBYTE128[128];
PMQCHAR	typedef MQCHAR MQPOINTER PMQCHAR;
PMQCHAR4	typedef MQCHAR4[4] MQPOINTER PMQCHAR4[4];
PMQCHAR8	typedef MQCHAR8[8] MQPOINTER PMQCHAR8[8];
PMQCHAR12	typedef MQCHAR12[12] MQPOINTER PMQCHAR12[12];
PMQCHAR20	typedef MQCHAR20[20] MQPOINTER PMQCHAR20[20];
PMQCHAR28	typedef MQCHAR28[28] MQPOINTER PMQCHAR28[28];
PMQCHAR32	typedef MQCHAR32[32] MQPOINTER PMQCHAR32[32];
PMQCHAR48	typedef MQCHAR48[48] MQPOINTER PMQCHAR48[48];
PMQCHAR64	typedef MQCHAR64[64] MQPOINTER PMQCHAR64[64];
PMQCHAR128	typedef MQCHAR128[128] MQPOINTER PMQCHAR128[128];
PMQCHAR256	typedef MQCHAR256[256] MQPOINTER PMQCHAR256[256];
PMQCHAR264	typedef MQCHAR264[264] MQPOINTER PMQCHAR264[264];
PMQCIH	typedef MQCIH MQPOINTER PMQCIH;
PMQCNO	typedef MQCNO MQPOINTER PMQCNO;
PMQDLH	typedef MQDLH MQPOINTER PMQDLH;
PMQFUNC	typedef void MQPOINTER PMQFUNC;

Elementary data types

Table 19. Elementary data types in C (continued)

Data type	Representation
PMQGMO	typedef MQGMO MQPOINTER PMQGMO;
PMQHCONFIG	typedef MQHCONFIG MQPOINTER PMQHCONFIG;
PMQHCONN	typedef MQHCONN MQPOINTER PMQHCONN;
PMQHOBJ	typedef MQHOBJ MQPOINTER PMQHOBJ;
PMQIIH	typedef MQIIH MQPOINTER PMQIIH;
PMQLONG	typedef MQLONG MQPOINTER PMQLONG;
PMQMD	typedef MQMD MQPOINTER PMQMD;
PMQMD1	typedef MQMD1[1] MQPOINTER PMQMD1[1];
PMQMDE	typedef MQMDE MQPOINTER PMQMDE;
PMQOD	typedef MQOD MQPOINTER PMQOD;
PMQPMO	typedef MQPMO MQPOINTER PMQPMO;
PMQPTR	typedef MQPTR MQPOINTER PMQPTR;
PMQRFH	typedef MQRFH MQPOINTER PMQRFH;
PMQRFH2	typedef MQRFH2[2] MQPOINTER PMQRFH2[2];
PMQRMH	typedef MQRMH MQPOINTER PMQRMH;
PMQTM	typedef MQTM MQPOINTER PMQTM;
PMQTM2	typedef MQTMC2[2] MQPOINTER PMQTM2[2];
PMQULONG	typedef MQULONG MQPOINTER PMQULONG;
PMQVOID	typedef void MQPOINTER PMQVOID;
PMQWIH	typedef MQWIH MQPOINTER PMQWIH;
PMQXQH	typedef MQXQH MQPOINTER PMQXQH;
PPMQBO	typedef PMQBO MQPOINTER PPMQBO;
PPMQBYTE	typedef PMQBYTE MQPOINTER PPMQBYTE;
PPMQCHAR	typedef PMQCHAR MQPOINTER PPMQCHAR;
PPMQCNO	typedef PMQCNO MQPOINTER PPMQCNO;
PPMQGMO	typedef PMQGMO MQPOINTER PPMQGMO;
PPMQHCONN	typedef PMQHCONN MQPOINTER PPMQHCONN;
PPMQHOBJ	typedef PMQHOBJ MQPOINTER PPMQHOBJ;
PPMQLONG	typedef PMQLONG MQPOINTER PPMQLONG;
PPMQMD	typedef PMQMD MQPOINTER PPMQMD;
PPMQOD	typedef PMQOD MQPOINTER PPMQOD;
PPMQPMO	typedef PMQPMO MQPOINTER PPMQPMO;
PPMQULONG	typedef PMQULONG MQPOINTER PPMQULONG;
PPMQVOID	typedef PMQVOID MQPOINTER PPMQVOID;

Where defined(MQ_64_BIT) means a 64 bit platform.

See “Data types” on page 20 for a description of the MQPOINTER macro variable.

COBOL declarations

Table 20. Elementary data types in COBOL

Data type	Representation
MQBYTE	PIC X
MQBYTE8	PIC X(8)
MQBYTE16	PIC X(16)
MQBYTE24	PIC X(24)
MQBYTE32	PIC X(32)
MQBYTE40	PIC X(40)
MQCHAR	PIC X
MQCHAR4	PIC X(4)
MQCHAR8	PIC X(8)
MQCHAR12	PIC X(12)
MQCHAR20	PIC X(20)
MQCHAR28	PIC X(28)
MQCHAR32	PIC X(32)
MQCHAR48	PIC X(48)
MQCHAR64	PIC X(64)
MQCHAR128	PIC X(128)
MQCHAR256	PIC X(256)
MQHCONN	PIC S9(9) BINARY
MQHOBJ	PIC S9(9) BINARY
MQLONG	PIC S9(9) BINARY
MQPTR	POINTER

PL/I declarations

PL/I is supported on z/OS, OS/2, VSE/ESA, and Windows systems.

Table 21. Elementary data types in PL/I

Data type	Representation
MQBYTE	char(1)
MQBYTE8	char(8)
MQBYTE16	char(16)
MQBYTE24	char(24)
MQBYTE32	char(32)
MQBYTE40	char(40)
MQCHAR	char(1)
MQCHAR4	char(4)
MQCHAR8	char(8)
MQCHAR12	char(12)

Elementary data types

Table 21. Elementary data types in PL/I (continued)

Data type	Representation
MQCHAR20	char(20)
MQCHAR28	char(28)
MQCHAR32	char(32)
MQCHAR48	char(48)
MQCHAR64	char(64)
MQCHAR128	char(128)
MQCHAR256	char(256)
MQHCONN	fixed bin(31)
MQHOBJ	fixed bin(31)
MQLONG	fixed bin(31)
MQPTR	pointer

System/390 assembler declarations

System/390 assembler is supported on z/OS only.

Table 22. Elementary data types in System/390 assembler

Data type	Representation
MQBYTE	DS XL1
MQBYTE8	DS XL8
MQBYTE16	DS XL16
MQBYTE24	DS XL24
MQBYTE32	DS XL32
MQBYTE40	DS XL40
MQCHAR	DS CL1
MQCHAR4	DS CL4
MQCHAR8	DS CL8
MQCHAR12	DS CL12
MQCHAR20	DS CL20
MQCHAR28	DS CL28
MQCHAR32	DS CL32
MQCHAR48	DS CL48
MQCHAR64	DS CL64
MQCHAR128	DS CL128
MQCHAR256	DS CL256
MQHCONN	DS F
MQHOBJ	DS F
MQLONG	DS F
MQPTR	DS F

TAL declarations

TAL is supported on Compaq NonStop Kernel only.

Table 23. Elementary data types in TAL

Data Type	Representation
MQBYTE	STRING
MQBYTE8	BEGIN STRING BYTE [0:7];END
MQBYTE16	BEGIN STRING BYTE [0:15];END
MQBYTE24	BEGIN STRING BYTE [0:23];END
MQBYTE32	BEGIN STRING BYTE [0:31];END
MQBYTE40	BEGIN STRING BYTE [0:39];END
MQCHAR	STRING
MQCHAR4	BEGIN STRING BYTE [0:3];END
MQCHAR8	BEGIN STRING BYTE [0:7]; END
MQCHAR12	BEGIN STRING BYTE [0:11];END
MQCHAR20	BEGIN STRING BYTE [0:19];END
MQCHAR28	BEGIN STRING BYTE [0:27];END
MQCHAR32	BEGIN STRING BYTE [0:31];END
MQCHAR48	BEGIN STRING BYTE [0:47];END
MQCHAR64	BEGIN STRING BYTE [0:63];END
MQCHAR128	BEGIN STRING BYTE [0:127];END
MQCHAR256	BEGIN STRING BYTE [0:255];END
MQHCONN	INT(32)
MQHOBJ	INT(32)
MQLONG	INT(32)
MQPTR	INT(32)

Visual Basic declarations

Visual Basic is supported on Windows systems only.

Table 24. Elementary data types in Visual Basic

Data type	Representation
MQBYTE	Byte
MQBYTE8	structure
MQBYTE16	structure
MQBYTE24	structure
MQBYTE32	structure
MQBYTE40	structure
MQCHAR	String*1
MQCHAR4	String*4
MQCHAR8	String*8
MQCHAR12	String*12

Elementary data types

Table 24. Elementary data types in Visual Basic (continued)

Data type	Representation
MQCHAR20	String*20
MQCHAR28	String*28
MQCHAR32	String*32
MQCHAR48	String*48
MQCHAR64	String*64
MQCHAR128	String*128
MQCHAR256	String*256
MQHCONN	Long
MQHOBJ	Long
MQLONG	Long
MQPTR	structure

|

Structure data types – introduction

This section introduces the structure data types used in the MQI. The structure data types themselves are described in subsequent chapters.

Summary

The following tables summarize the structure data types used in the MQI:

Table 25. Structure data types used on MQI calls (or exit functions):

Structure	Description	Calls where used
MQACH	API exit chain header	
MQAIR	Authentication information record	MQCONN
MQAXC	API exit context	
MQAXP	API exit parameter	
MQBO	Begin options	MQBEGIN
MQCNO	Connect options	MQCONN
MQGMO	Get-message options	MQGET
MQMD	Message descriptor	MQGET, MQPUT, MQPUT1
MQOD	Object descriptor	MQOPEN, MQPUT1
MQOR	Object record	MQOPEN, MQPUT1
MQPMO	Put-message options	MQPUT, MQPUT1
MQPMR	Put-message record	MQPUT, MQPUT1
MQRR	Response record	MQOPEN, MQPUT, MQPUT1
MQSCO	SSL configuration options	MQCONN

Table 26. Structure data types used in message data:

Structure	Description
MQCIH	CICS information header
MQDH	Distribution header
MQDLH	Dead letter (undelivered message) header
MQIIH	IMS™ information header
MQMDE	Message descriptor extension
MQRFH	Rules and formatting header
MQRFH2	Rules and formatting header 2
MQRMH	Reference message header
MQTM	Trigger message
MQTMC2	Trigger message (character format 2)
MQWIH	Work Information header
MQXQH	Transmission queue header

Note: The MQDXP structure (data conversion exit parameter) is described in Appendix F, “Data conversion”, on page 581, together with the associated data conversion calls.

Rules for structure data types

Rules for structure data types

Programming languages vary in their level of support for structures, and certain rules and conventions are adopted in order to allow the MQI structures to be mapped consistently in each programming language:

1. Structures must be aligned on their natural boundaries.
 - Most MQI structures require 4-byte alignment.
 - On OS/400, structures containing pointers require 16-byte alignment; these are: MQCNO, MQOD, MQPMO.
2. Each field in a structure must be aligned on its natural boundary.
 - Fields with data types that equate to MQLONG must be aligned on 4-byte boundaries.
 - Fields with data types that equate to MQPTR must be aligned on 16-byte boundaries on OS/400, and 4-byte boundaries in other environments.
 - Other fields are aligned on 1-byte boundaries.
3. The length of a structure must be a multiple of its boundary alignment.
 - Most MQI structures have lengths that are multiples of 4 bytes.
 - On OS/400, structures containing pointers have lengths that are multiples of 16 bytes.
4. Where necessary, padding bytes or fields must be added to ensure compliance with the above rules.

Conventions used in the descriptions

The description of each structure data type includes:

- An overview of the purpose and use of the structure
- Descriptions of the fields in the structure, in a form that is independent of the programming language
- Examples of how the structure is declared, in each of the supported programming languages

The description of each structure data type contains the following sections:

Structure name

The name of the structure, followed by a summary of the fields in the structure.

Overview

A brief description of the purpose and use of the structure.

Fields Descriptions of the fields. For each field, the name of the field is followed by its elementary data type in parentheses (). In text, field names are shown using an italic typeface; for example: *Version*.

There is also a description of the purpose of the field, together with a list of any values that the field can take. Names of constants are shown in uppercase; for example, MQGMO_STRUC_ID. A set of constants having the same prefix is shown using the * character, for example: MQIA_*.

In the descriptions of the fields, the following terms are used:

input You supply information in the field when you make a call.

output

The queue manager returns information in the field when the call completes or fails.

input/output

You supply information in the field when you make a call, and the queue manager changes the information when the call completes or fails.

Initial values

A table showing the initial values for each field in the data definition files supplied with the MQI.

C declaration

Typical declaration of the structure in C.

COBOL declaration

Typical declaration of the structure in COBOL.

PL/I declaration

Typical declaration of the structure in PL/I.

System/390 assembler declaration

Typical declaration of the structure in System/390 assembler language.

Visual Basic declaration

Typical declaration of the structure in Visual basic.

C programming

This section contains information to help you use the MQI from the C programming language.

Header files

Header files are provided to assist with the writing of C application programs that use the MQI. These header files are summarized in Table 27.

Table 27. C header files

File	Contents
CMQC	Function prototypes, data types, and named constants for the main MQI
CMQXC	Function prototypes, data types, and named constants for the data conversion exit

To improve the portability of applications, it is recommended that the name of the header file should be coded in lowercase on the **#include** preprocessor directive:

```
#include "cmqc.h"
```

Functions

- Parameters that are *input-only* and of type MQHCONN, MQHOBJ, or MQLONG are passed by value.
- All other parameters are passed by address.

Not all parameters that are passed by address need to be specified every time a function is invoked. Where a particular parameter is not required, a null pointer can be specified as the parameter on the function invocation, in place of the address of the parameter data. Parameters for which this is possible are identified in the call descriptions.

C programming

No parameter is returned as the value of the function; in C terminology, this means that all functions return **void**.

The attributes of the function are defined by the MQENTRY macro variable; the value of this macro variable depends on the environment.

Parameters with undefined data type

The MQGET, MQPUT, and MQPUT1 functions each have one parameter that has an undefined data type, namely the *Buffer* parameter. This parameter is used to send and receive the application's message data.

Parameters of this sort are shown in the C examples as arrays of MQBYTE. It is perfectly valid to declare the parameters in this way, but it is usually more convenient to declare them as the particular structure which describes the layout of the data in the message. The actual function parameter is declared as a pointer-to-void, and so the address of any sort of data can be specified as the parameter on the function invocation.

Data types

All data types are defined by means of the C **typedef** statement. For each data type, the corresponding pointer data type is also defined. The name of the pointer data type is the name of the elementary or structure data type prefixed with the letter "P" to denote a pointer. The attributes of the pointer are defined by the MQPOINTER macro variable; the value of this macro variable depends on the environment. The following illustrates how pointer datatypes are declared:

```
#define MQPOINTER *          /* depends on environment */
...
typedef MQLONG MQPOINTER PMQLONG; /* pointer to MQLONG */
typedef MQMD MQPOINTER PMQMD; /* pointer to MQMD */
```

Manipulating binary strings

Strings of binary data are declared as one of the MQBYTEn data types. Whenever fields of this type are copied, compared, or set, the C functions **memcpy**, **memcmp**, or **memset** should be used; for example:

```
#include <string.h>
#include "cmqc.h"

MQMD MyMsgDesc;

memcpy(MyMsgDesc.MsgId,          /* set "MsgId" field to nulls */
       MQMI_NONE,              /* ...using named constant */
       sizeof(MyMsgDesc.MsgId));

memset(MyMsgDesc.CorrelId,      /* set "CorrelId" field to nulls */
       0x00,                   /* ...using a different method */
       sizeof(MQBYTE24));
```

Do not use the string functions **strcpy**, **strcmp**, **strncpy**, or **strncmp**, because these do not work correctly for data declared with the MQBYTEn data types.

Manipulating character strings

When the queue manager returns character data to the application, the queue manager always pads the character data with blanks to the defined length of the field; the queue manager *does not* return null-terminated strings. Therefore, when copying, comparing, or concatenating such strings, the string functions **strncpy**, **strncmp**, or **strncat** should be used.

Do not use the string functions, which require the string to be terminated by a null (**strcpy**, **strcmp**, **strcat**). Also, do not use the function **strlen** to determine the length of the string; use instead the **sizeof** function to determine the length of the field.

Initial values for structures

The header files define various macro variables that can be used to provide initial values for the MQ structures when instances of those structures are declared. These macro variables have names of the form MQxxx_DEFAULT, where MQxxx represents the name of the structure. They are used in the following way:

```
MQMD   MyMsgDesc = {MQMD_DEFAULT};
MQPMO  MyPutOpts = {MQPMO_DEFAULT};
```

For some character fields (for example, the *StrucId* fields which occur in most structures, or the *Format* field which occurs in MQMD), the MQI defines particular values that are valid. For each of the valid values, *two* macro variables are provided:

- One macro variable defines the value as a string whose length excluding the implied null matches exactly the defined length of the field. For example, for the *Format* field in MQMD the following macro variable is provided (the symbol “b” represents a blank character):

```
#define MQFMT_STRING "MQSTRbbb"
```

Use this form with the **memcpy** and **memcmp** functions.

- The other macro variable defines the value as an array of characters; the name of this macro variable is the name of the string form suffixed with **_ARRAY**. For example:

```
#define MQFMT_STRING_ARRAY 'M','Q','S','T','R','b','b','b'
```

Use this form to initialize the field when an instance of the structure is declared with values different from those provided by the MQMD_DEFAULT macro variable.²

Initial values for dynamic structures

When a variable number of instances of a structure is required, the instances are usually created in main storage obtained dynamically using the **calloc** or **malloc** functions. To initialize the fields in such structures, the following technique is recommended:

1. Declare an instance of the structure using the appropriate MQxxx_DEFAULT macro variable to initialize the structure. This instance becomes the “model” for other instances:

```
MQMD Model = {MQMD_DEFAULT}; /* declare model instance */
```

The **static** or **auto** keywords can be coded on the declaration in order to give the model instance static or dynamic lifetime, as required.

2. Use the **calloc** or **malloc** functions to obtain storage for a dynamic instance of the structure:

```
PMQMD Instance;
Instance = malloc(sizeof(MQMD)); /* get storage for dynamic instance */
```

3. Use the **memcpy** function to copy the model instance to the dynamic instance:

```
memcpy(Instance,&Model,sizeof(MQMD)); /* initialize dynamic instance */
```

2. This is not always necessary; in some environments the string form of the value can be used in both situations. However, the array form is recommended for declarations, since this is required for compatibility with the C++ programming language.

C programming

Use from C++

For the C++ programming language, the header files contain the following additional statements that are included only when a C++ compiler is used:

```
#ifdef __cplusplus
extern "C" {
#endif

/* rest of header file */

#ifdef __cplusplus
}
#endif
```

Notational conventions

The later sections in this book show how the functions should be invoked and parameters declared. In some cases, the parameters are arrays whose size is not fixed. For these, a lowercase “n” is used to represent a numeric constant. When you code the declaration for that parameter, the “n” must be replaced by the numeric value required.

COBOL programming

This section contains information to help you use the MQI from the COBOL programming language.

COPY files

Various COPY files are provided to assist with the writing of COBOL application programs that use the MQI. There are two files containing named constants, and two files for each of the structures.

Each structure is provided in two forms – a form with initial values, and a form without:

- The structures with initial values can be used in the **WORKING-STORAGE SECTION** of a COBOL program, and are contained in COPY files which have names suffixed with the letter “V” (mnemonic for “Values”).
- The structures without initial values can be used in the **LINKAGE SECTION** of a COBOL program, and are contained in COPY files which have names suffixed with the letter “L” (mnemonic for “Linkage”).

For the COPY files for which there are multiple versions, be sure to use the appropriate version for your purpose. For example, for the message descriptor structure, use:

CMQMD1x

COPY files for compatibility with code compiled on versions of MQSeries before WebSphere MQ Version 5.3.

CMQMD2x

COPY files when you want to write new code to exploit group or segmented message functions.

Notes:

1. The CMQMDx COPY files are retained for compatibility with earlier versions of MQSeries.
2. Take great care when mixing programs that use structures of different versions. In particular, be careful how you pass the message descriptor (MQMD)

structure through other routines to the application layer where the MQPUT or MQGET is executed. You might need to change intermediate programs to cope with the increased size of the MQMD structure, or code the application to be aware of the version of MQMD being passed.

The COPY files are summarized in Table 28. Note that not all of the files listed are available in all environments.

Table 28. COBOL COPY files

File (with initial values)	File (without initial values)	Contents
CMQAIRV	CMQAIRL	Authentication information record
CMQBOV	CMQBOL	Begin options structure
CMQCIHV	CMQCIHL	CICS information header structure
CMQCNOV	CMQCNOL	Connect options structure
CMQDHV	CMQDHL	Distribution header structure
CMQDLHV	CMQDLHL	Dead letter header structure
CMQDXPV	CMQDXPL	Data conversion exit parameter structure
CMQGMOV	CMQGMOL	Get message options structure
CMQIIHV	CMQIIHL	IMS information header structure
CMQMDV	CMQMDL	Message descriptor structure
CMQMDEV	CMQMDEL	Message descriptor extension structure
CMQMD1V	CMQMD1L	Message descriptor structure version 1
CMQMD2V	CMQMD2L	Message descriptor structure version 2
CMQODV	CMQODL	Object descriptor structure
CMQORV	CMQORL	Object record structure
CMQPMOV	CMQPMOL	Put message options structure
CMQRFHV	CMQRFHL	Rules and formatting header structure
CMQRFH2V	CMQRFH2L	Rules and formatting header structure version 2
CMQRMHV	CMQRMHL	Reference message header structure
CMQRRV	CMQRRL	Response record structure
CMQSCOV	CMQSCOL	SSL configuration options
CMQTMV	CMQTML	Trigger message structure
CMQTMCV	CMQTMCL	Trigger message structure (character format)
CMQTM2V	CMQTM2L	Trigger message structure (character format) version 2
CMQWIHV	CMQWIHL	Work information header structure
CMQXQHV	CMQXQHL	Transmission queue header structure
CMQV	–	Named constants for main MQI
CMQXV	–	Named constants for data conversion exit

Structures

In the COPY file, each structure declaration begins with a level-10 item; this enables several instances of the structure to be declared, by coding the level-01

COBOL programming

declaration and then using the **COPY** statement to copy in the remainder of the structure declaration. To reference the appropriate instance, the **IN** keyword can be used:

```
* Declare two instances of MQMD
01 MY-MQMD.
   COPY CMQMDV.
01 MY-OTHER-MQMD.
   COPY CMQMDV.
*
* Set MSGTYPE field in MY-OTHER-MQMD
  MOVE MQMT-REQUEST TO MQMD-MSGTYPE IN MY-OTHER-MQMD.
```

The structures should be aligned on appropriate boundaries. If the **COPY** statement is used to include a structure following an item which is not the level-01 item, try to ensure that the structure begins at the appropriate offset from the start of the level-01 item; failure to do this may result in a performance degradation or other problems. Most MQI structures require 4-byte alignment; the exceptions to this are MQCNO, MQOD, and MQPMO, which require 16-byte alignment on OS/400.

In this book, the names of fields in structures are shown without a prefix. In COBOL, the field names are prefixed with the name of the structure followed by a hyphen. However, if the structure name ends with a numeric digit, indicating that the structure is a second or later version of the original structure, the numeric digit is *omitted* from the prefix. Field names in COBOL are shown in uppercase (although lowercase or mixed case can be used if required). For example, the field *MsgType* described in Chapter 10, "MQMD – Message descriptor", on page 141 becomes MQMD-MSGTYPE in COBOL.

The V-suffix structures are declared with initial values for all of the fields, and so it is necessary to set only those fields where the value required is different from the supplied initial value.

Pointers

Some structures need to address optional data that may be discontinuous with the structure. For example, the MQOR and MQRR records addressed by the MQOD structure are like this. To address this optional data, the structures contain fields that are declared with the pointer data type. However, COBOL does not support the pointer data type in all environments. Because of this, the optional data can also be addressed using fields which contain the offset of the data from the start of the structure.

If an application is intended to be portable between environments, the application designer should ascertain whether the pointer data type is available in all of the intended environments. If it is not, the application should address the optional data using the offset fields instead of the pointer fields.

In those environments where pointers are not supported, the pointer fields are declared as byte strings of the appropriate length, with the initial value being the all-null byte string. This initial value should not be altered if the offset fields are being used.

Named constants

In this book, the names of constants are shown containing the underscore character () as part of the name. In COBOL, the hyphen character (-) must be used in place of the underscore.

Constants which have character-string values use the single-quote character as the string delimiter ('). In some environments it may be necessary to specify an appropriate compiler option to cause the compiler to accept the single quote as the string delimiter in place of the double quote.

The named constants are declared in the COPY files as level-10 items. To use the constants, the level-01 item must be declared explicitly, and then the **COPY** statement used to copy in the declarations of the constants:

```
* Declare a structure to hold the constants
01 MY-MQ-CONSTANTS.
   COPY CMQV.
```

The above method causes the constants to occupy storage in the program even if they are not referenced. If the constants are included in many separate programs within the same run unit, multiple copies of the constants will exist; this consumes main storage unnecessarily. This can be avoided by using one of the following techniques:

- Add the **GLOBAL** clause to the level-01 declaration:

```
* Declare a global structure to hold the constants
01 MY-MQ-CONSTANTS GLOBAL.
   COPY CMQV.
```

This causes storage to be allocated for only *one* set of constants within the run unit; the constants, however, can be referenced by *any* program within the run unit, not just the program that contains the level-01 declaration.

Note: The **GLOBAL** clause is not supported in all environments.

- Manually copy into each program only those constants that are referenced by that program; do not use the **COPY** statement to copy all of the constants into the program.

Notational conventions

The later sections in this book show how the calls should be invoked and parameters declared. In some cases, the parameters are tables or character strings whose size is not fixed. For these, a lowercase “n” is used to represent a numeric constant. When you code the declaration for that parameter, the “n” must be replaced by the numeric value required.

PL/I programming

This section contains information to help you use the MQI from the PL/I programming language.

INCLUDE files

Two **INCLUDE** files are provided to assist with the writing of PL/I application programs that use the MQI. There is one **INCLUDE** file containing the structures and named constants, and one containing the entry-point declarations. These files are summarized in Table 29.

Table 29. PL/I **INCLUDE** files

File	Contents
CMQEPP	Entry points
CMQP	Structures and named constants

PL/I programming

To improve the portability of applications, it is recommended that the names of the `INCLUDE` files should be coded in lowercase on the `%include` compiler directive:

```
%include syslib(cmqp);
%include syslib(cmqep);
```

Structures

Structures are declared with the **BASED** attribute, and so do not occupy any storage unless the program declares one or more instances of a structure.

An instance of a structure can be declared by using the **LIKE** attribute:

```
%include syslib(cmqp);
%include syslib(cmqep);

dcl 1 my_mqmd          like MQMD; /* one instance */
dcl 1 my_other_mqmd   like MQMD; /* another one */
```

The structure fields are declared with the **INITIAL** attribute. When the **LIKE** attribute is used to declare an instance of a structure, that instance inherits the initial values defined for that structure. Thus it is necessary to set only those fields where the value required is different from the initial value supplied.

PL/I is not sensitive to case, and so the names of calls, structure fields, and constants can be coded in upper, lower, or mixed case.

Named constants

The named constants are declared as macro variables. As a result, named constants that are not referenced by the program do not occupy any storage in the compiled procedure. However, the compiler option that causes the source to be processed by the macro preprocessor must be specified when the program is compiled.

All of the macro variables are character variables, even the ones that represent numeric values. Although this may seem counter-intuitive, it does not result in any data-type conflict after the macro variables have been substituted by the macro processor:

```
%dcl MQMD_STRUC_ID char;
%MQMD_STRUC_ID = 'MQMD';

%dcl MQMD_VERSION_1 char;
%MQMD_VERSION_1 = '1';
```

Notational conventions

The later sections in this book show how the calls should be invoked and parameters declared. In some cases, the parameters are arrays or character strings whose size is not fixed. For these, a lowercase “n” is used to represent a numeric constant. When you code the declaration for that parameter, the “n” must be replaced by the numeric value required.

System/390 assembler programming

This section contains information to help you use the MQI from the System/390 Assembler programming language.

Macros

Various macros are provided to assist with the writing of assembler application programs that use the MQI. There are two macros for named constants, and one

macro for each of the structures. These files are summarized in Table 30.

Table 30. Assembler macros

File	Contents
CMQA	Named constants (“equates”) for main MQI
CMQCIHA	CICS information header structure
CMQCNOA	Connect options structure
CMQDLHA	Dead letter header structure
CMQDXPA	Data conversion exit parameter structure
CMQGMOA	Get message options structure
CMQIIHA	IMS information header structure
CMQMDA	Message descriptor structure
CMQMDEA	Message descriptor extension structure
CMQODA	Object descriptor structure
CMQPMOA	Put message options structure
CMQRFHA	Rules and formatting header structure
CMQRFH2A	Rules and formatting header structure version 2
CMQRMHA	Reference message header structure
CMQTMA	Trigger message structure
CMQTM2A	Trigger message structure (character format) version 2
CMQVERA	Structure version control
CMQWIHA	Work information header structure
CMQXA	Named constants for data conversion exit
CMQXPA	API crossing exit parameter structure
CMQXQHA	Transmission queue header structure

Structures

The structures are generated by macros that have various parameters to control the action of the macro. These parameters are described in the following sections.

From time to time new versions of the MQ structures are introduced. The additional fields in a new version can cause a structure that previously was smaller than 256 bytes to become larger than 256 bytes. Because of this, it is recommended that assembler instructions that are intended to copy an MQ structure, or to set an MQ structure to nulls, should be written to work correctly with structures that may be larger than 256 bytes. Alternatively, the **DCLVER** macro parameter or **CMQVERA** macro can be used with the **VERSION** parameter to cause a specific version of the structure to be declared (see below).

Specifying the name of the structure

To allow more than one instance of a structure to be declared, the macro prefixes the name of each field in the structure with a user-specifiable string and an underscore. The string used is the label specified on the invocation of the macro. If no label is specified, the name of the structure is used to construct the prefix:

```
* Declare two object descriptors
           CMQODA ,           Prefix used="MQOD_" (the default)
MY_MQOD CMQODA ,           Prefix used="MY_MQOD_"
```

System/390 assembler programming

The structure declarations shown in this book use the default prefix.

Specifying the form of the structure

Structure declarations can be generated by the macro in one of two forms, controlled by the **DSECT** parameter:

DSECT=YES

An assembler **DSECT** instruction is used to start a new data section; the structure definition immediately follows the **DSECT** statement. The label on the macro invocation is used as the name of the data section; if no label is specified, the name of the structure is used.

DSECT=NO

Assembler **DC** instructions are used to define the structure at the current position in the routine. The fields are initialized with values, which can be specified by coding the relevant parameters on the macro invocation. Fields for which no values are specified on the macro invocation are initialized with default values.

The value specified must be uppercase. If the **DSECT** parameter is not specified, **DSECT=NO** is assumed.

Controlling the version of the structure

By default, the macros always declare the most recent version of each structure. Although you can use the **VERSION** macro parameter to specify a value for the *Version* field in the structure, that parameter defines the initial value for the *Version* field, and does not control the version of the structure actually declared. To control the version of the structure that is declared, use the **DCLVER** parameter:

DCLVER=CURRENT

The version declared is the current (most recent) version.

DCLVER=SPECIFIED

The version declared is the version specified by the **VERSION** parameter. If the **VERSION** parameter is omitted, the default is version 1.

If the **VERSION** parameter is specified, the value must be a self-defining numeric constant, or the named constant for the version required (for example, **MQCNO_VERSION_3**). If some other value is specified, the structure is declared as if **DCLVER=CURRENT** had been specified, even if the value of **VERSION** resolves to a valid value.

The value specified must be uppercase. If the **DCLVER** parameter is not specified, the value used is taken from the **MQDCLVER** global macro variable. This variable can be set by means of the **CMQVERA** macro (see below).

Declaring one structure embedded within another

To declare one structure as a component of another structure, the **NESTED** parameter should be used:

NESTED=YES

The structure declaration is nested within another.

NESTED=NO

The structure declaration is not nested within another.

The value specified must be uppercase. If the **NESTED** parameter is not specified, **NESTED=NO** is assumed.

Specifying initial values for fields

The value to be used to initialize a field in a structure can be specified by coding the name of that field (without the prefix) as a parameter on the macro invocation, accompanied by the value required. For example, to declare a message-descriptor structure with the *MsgType* field initialized with `MQMT_REQUEST`, and the *ReplyToQ* field initialized with the string `"MY_REPLY_TO_QUEUE"`, the following could be used:

```
MY_MQMD  CMQMDA  MSGTYPE=MQMT_REQUEST,           X
          REPLYTOQ=MY_REPLY_TO_QUEUE
```

If a named constant (equate) is specified as a value on the macro invocation, the `CMQA` macro must be used in order to define the named constant. Values which are character strings must not be enclosed in single quotes.

Controlling the listing

The appearance of the structure declaration in the assembler listing can be controlled by means of the `LIST` parameter:

`LIST=YES`

The structure declaration appears in the assembler listing.

`LIST=NO`

The structure declaration does not appear in the assembler listing.

The value specified must be uppercase. If the `LIST` parameter is not specified, `LIST=NO` is assumed.

CMQVERA macro

This macro allows you to set the default value to be used for the `DCLVER` parameter on the structure macros. The value specified by `CMQVERA` is used by the structure macro only if the `DCLVER` parameter is not specified on the invocation of the structure macro. The default value is set by coding the `CMQVERA` macro with the `DCLVER` parameter:

`DCLVER=CURRENT`

The default version is set to the current (most recent) version.

`DCLVER=SPECIFIED`

The default version is set to the version specified by the `VERSION` parameter.

The `DCLVER` parameter must be specified, and the value must be uppercase. The value set by `CMQVERA` remains the default value until the next invocation of `CMQVERA`, or the end of the assembly. If `CMQVERA` is not used, the default is `DCLVER=CURRENT`.

Notational conventions

The later sections in this book show how the calls should be invoked and parameters declared. In some cases, the parameters are arrays or character strings whose size is not fixed. For these, a lowercase `"n"` is used to represent a numeric constant. When you code the declaration for that parameter, the `"n"` must be replaced by the numeric value required.

Visual Basic programming

This section contains information to help you use the MQI from the Visual Basic programming language.

Visual Basic Programming

Header files

Header (or form) files are provided to assist with the writing of Visual Basic application programs that use the MQI. These header files are summarized in Table 31.

Table 31. Visual Basic header files

File	Contents
CMQB	Call declarations, data types, and named constants for the main MQI.

In a default installation, the module files (.BAS) are supplied in the \Program Files\MQSeries for Windows NT\Samples\VB\Include subdirectory.

Parameters of the MQI calls

Parameters that are *input-only* and of type MQHCONN, MQHOBJ, or MQLONG are passed by value; all other parameters are passed by address.

Initial values for structures

The supplied header files define various subroutines that can be invoked to initialize the MQ structures with the default values. These subroutines have names of the form **MQxxx_DEFAULTS**, where **MQxxx** represents the name of the structure. They are used in the following way:

```
MQMD_DEFAULTS (MyMsgDesc)      'Initialize message descriptor'  
MQPMO_DEFAULTS (MyPutOpts)    'Initialize put-message options'
```

There is also a subroutine called **MQ_SETDEFAULTS**, that you call at the start of a program to ensure that various default constants are set up properly.

MQ_SETDEFAULTS should be called before any other MQSeries calls, and you are recommended to put this subroutine in the Load procedure of the start up form. For example:

```
Private Sub Form_Load()  
    ' Set up default constants  
    MQ_SETDEFAULTS  
End Sub
```

Notational conventions

The later sections in this book show how the functions should be invoked and parameters declared. In some cases, the parameters are arrays whose size is not fixed. For these, a lowercase “n” is used to represent a numeric constant. When you code the declaration for that parameter, the “n” must be replaced by the numeric value required.

Chapter 2. MQAIR – Authentication information record

The following table summarizes the fields in the structure.

Table 32. Fields in MQAIR

Field	Description	Page
<i>StrucId</i>	Structure identifier	33
<i>Version</i>	Structure version number	34
<i>AuthInfoType</i>	Type of authentication information	32
<i>AuthInfoConnName</i>	Connection name of LDAP CRL server	31
<i>LDAPUserNamePtr</i>	Address of LDAP user name	33
<i>LDAPUserNameOffset</i>	Offset of LDAP user name from start of MQSCO	32
<i>LDAPUserNameLength</i>	Length of LDAP user name	32
<i>LDAPPASSWORD</i>	Password to access LDAP server	32

Overview

Availability: AIX, HP-UX, Solaris, Linux and Windows clients, apart from Windows 98 clients.

Purpose: The MQAIR structure allows an application running as a WebSphere MQ client to specify information about an authenticator that is to be used for the client connection. The structure is an input parameter on the MQCONN call.

Character set and encoding: Data in MQAIR must be in the character set and encoding of the local queue manager; these are given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE, respectively.

Fields

The MQAIR structure contains the following fields; the fields are described in **alphabetic order**:

AuthInfoConnName (MQCHAR264)

Connection name of CRL LDAP server.

This is either the host name or the network address of a host on which the LDAP server is running. This can be followed by an optional port number, enclosed in parentheses. The default port number is 389.

If the value is shorter than the length of the field, the value must be terminated by a null character, or padded with blanks to the length of the field. If the value is not valid, the call fails with reason code MQRC_AUTH_INFO_CONN_NAME_ERROR.

This is an input field. The length of this field is given by MQ_AUTH_INFO_CONN_NAME_LENGTH. The initial value of this field is the null string in C, and blank characters in other programming languages.

MQAIR – AuthInfoType field

AuthInfoType (MQLONG)

Type of authentication information.

This specifies the type of authentication information contained in the record. The value must be:

MQAIT_CRL_LDAP

Certificate revocation using LDAP server.

If the value is not valid, the call fails with reason code MQRC_AUTH_INFO_TYPE_ERROR.

This is an input field. The initial value of this field is MQAIT_CRL_LDAP.

LDAPPassword (MQCHAR32)

Password to access LDAP server.

This is the password needed to access the LDAP CRL server. If the value is shorter than the length of the field, the value must be terminated by a null character, or padded with blanks to the length of the field.

If the LDAP server does not require a password, or no LDAP user name is specified, *LDAPPassword* must be null or blank. If no LDAP user name is specified and *LDAPPassword* is not null or blank, the call fails with reason code MQRC_LDAP_PASSWORD_ERROR.

This is an input field. The length of this field is given by MQ_LDAP_PASSWORD_LENGTH. The initial value of this field is the null string in C, and blank characters in other programming languages.

LDAPUserNameLength (MQLONG)

Length of LDAP user name.

This is the length in bytes of the LDAP user name addressed by the *LDAPUserNamePtr* or *LDAPUserNameOffset* field. The value must be in the range zero through MQ_DISTINGUISHED_NAME_LENGTH. If the value is not valid, the call fails with reason code MQRC_LDAP_USER_NAME_LENGTH_ERR.

If the LDAP server involved does not require a user name, this field should be set to zero.

This is an input field. The initial value of this field is 0.

LDAPUserNameOffset (MQLONG)

Offset of LDAP user name from start of MQAIR structure.

This is the offset in bytes of the LDAP user name from the start of the MQAIR structure. The offset can be positive or negative. The field is ignored if *LDAPUserNameLength* is zero.

Either *LDAPUserNamePtr* or *LDAPUserNameOffset* can be used to specify the LDAP user name, but not both; see the description of the *LDAPUserNamePtr* field for details.

This is an input field. The initial value of this field is 0.

LDAPUserNamePtr (PMQCHAR)

Address of LDAP user name.

This is the LDAP user name. It consists of the Distinguished Name of the user who is attempting to access the LDAP CRL server. If the value is shorter than the length specified by *LDAPUserNameLength*, the value must be terminated by a null character, or padded with blanks to the length *LDAPUserNameLength*. The field is ignored if *LDAPUserNameLength* is zero.

The LDAP user name can be provided in one of two ways:

- By using the pointer field *LDAPUserNamePtr*

In this case, the application can declare a string that is separate from the MQAIR structure, and set *LDAPUserNamePtr* to the address of the string.

Using *LDAPUserNamePtr* is recommended for programming languages that support the pointer data type in a fashion that is portable to different environments (for example, the C programming language).

- By using the offset field *LDAPUserNameOffset*

In this case, the application should declare a compound structure containing the MQSCO structure followed by the array of MQAIR records followed by the LDAP user name strings, and set *LDAPUserNameOffset* to the offset of the appropriate name string from the start of the MQAIR structure. Care must be taken to ensure that this value is correct, and has a value that can be accommodated within an MQLONG (the most restrictive programming language is COBOL, for which the valid range is -999 999 999 through +999 999 999).

Using *LDAPUserNameOffset* is recommended for programming languages that do not support the pointer data type, or that implement the pointer data type in a fashion that may not be portable to different environments (for example, the COBOL programming language).

Whichever technique is chosen, only one of *LDAPUserNamePtr* and *LDAPUserNameOffset* can be used; the call fails with reason code MQRC_LDAP_USER_NAME_ERROR if both are nonzero.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise.

Note: On platforms where the programming language does not support the pointer datatype, this field is declared as a byte string of the appropriate length.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQAIR_STRUC_ID

Identifier for authentication information record.

For the C programming language, the constant MQAIR_STRUC_ID_ARRAY is also defined; this has the same value as MQAIR_STRUC_ID, but is an array of characters instead of a string.

MQAIR – Version field

This is always an input field. The initial value of this field is MQAIR_STRUC_ID.

Version (MQLONG)

Structure version number.

The value must be:

MQAIR_VERSION_1

Version-1 authentication information record.

The following constant specifies the version number of the current version:

MQAIR_CURRENT_VERSION

Current version of authentication information record.

This is always an input field. The initial value of this field is MQAIR_VERSION_1.

Initial values and language declarations

Table 33. Initial values of fields in MQAIR

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQAIR_STRUC_ID	'AIRb'
<i>Version</i>	MQAIR_CURRENT_VERSION	1
<i>AuthInfoType</i>	MQAIT_CRL_LDAP	1
<i>AuthInfoConnName</i>	None	Null string or blanks
<i>LDAPUserNamePtr</i>	None	Null pointer or null bytes
<i>LDAPUserNameOffset</i>	None	0
<i>LDAPUserNameLength</i>	None	0
<i>LDAPPASSWORD</i>	None	Null string or blanks

Notes:

1. The symbol 'b' represents a single blank character.
2. In the C programming language, the macro variable MQAIR_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQAIR MyAIR = {MQAIR_DEFAULT};
```

C declaration

```
typedef struct tagMQAIR MQAIR;
struct tagMQAIR {
    MQCHAR4    StrucId;           /* Structure identifier */
    MQLONG     Version;          /* Structure version number */
    MQLONG     AuthInfoType;     /* Type of authentication
                                information */
    MQCHAR264  AuthInfoConnName; /* Connection name of CRL LDAP
                                server */
    PMQCHAR    LDAPUserNamePtr;  /* Address of LDAP user name */
    MQLONG     LDAPUserNameOffset; /* Offset of LDAP user name from start
```

MQAIR – Language declarations

```

|                                     of MQAIR structure */
| MQLONG      LDAPUserNameLength; /* Length of LDAP user name */
| MQCHAR32    LDAPPASSWORD;      /* Password to access LDAP server */
| };

```

COBOL declaration

```

| ** MQAIR structure
| 10 MQAIR.
| ** Structure identifier
| 15 MQAIR-STRUCID      PIC X(4).
| ** Structure version number
| 15 MQAIR-VERSION     PIC S9(9) BINARY.
| ** Type of authentication information
| 15 MQAIR-AUTHINFOTYPE PIC S9(9) BINARY.
| ** Connection name of CRL LDAP server
| 15 MQAIR-AUTHINFOCONNNAME PIC X(264).
| ** Address of LDAP user name
| 15 MQAIR-LDAPUSERNAMEPTR POINTER.
| ** Offset of LDAP user name from start of MQAIR structure
| 15 MQAIR-LDAPUSERNAMEOFFSET PIC S9(9) BINARY.
| ** Length of LDAP user name
| 15 MQAIR-LDAPUSERNAMELENGTH PIC S9(9) BINARY.
| ** Password to access LDAP server
| 15 MQAIR-LDAPPASSWORD PIC X(32).

```

PL/I declaration

```

| dc1
| 1 MQAIR based,
| 3 StrucId      char(4), /* Structure identifier */
| 3 Version      fixed bin(31), /* Structure version number */
| 3 AuthInfoType fixed bin(31), /* Type of authentication
|                               information */
| 3 AuthInfoConnName char(264), /* Connection name of CRL LDAP
|                               server */
| 3 LDAPUserNamePtr pointer, /* Address of LDAP user name */
| 3 LDAPUserNameOffset fixed bin(31), /* Offset of LDAP user name from
|                               start of MQAIR structure */
| 3 LDAPUserNameLength fixed bin(31), /* Length of LDAP user name */
| 3 LDAPPASSWORD char(32); /* Password to access LDAP
|                               server */

```

Visual Basic declaration

```

| Type MQAIR
| StrucId      As String*4 'Structure identifier'
| Version      As Long      'Structure version number'
| AuthInfoType As Long      'Type of authentication information'
| AuthInfoConnName As String*264 'Connection name of CRL LDAP server'
| LDAPUserNamePtr As MQPTR   'Address of LDAP user name'
| LDAPUserNameOffset As Long  'Offset of LDAP user name from start'
|                               'of MQAIR structure'
| LDAPUserNameLength As Long  'Length of LDAP user name'
| LDAPPASSWORD As String*32  'Password to access LDAP server'
| End Type

```

MQAIR – Language declarations

Chapter 3. MQBO – Begin options

The following table summarizes the fields in the structure.

Table 34. Fields in MQBO

Field	Description	Page
<i>StrucId</i>	Structure identifier	37
<i>Version</i>	Structure version number	38
<i>Options</i>	Options that control the action of MQBEGIN	37

Overview

Availability: AIX, HP-UX, OS/2, OS/400, Solaris, Linux, Windows; not available for WebSphere MQ clients.

Purpose: The MQBO structure allows the application to specify options relating to the creation of a unit of work. The structure is an input/output parameter on the MQBEGIN call.

Character set and encoding: Data in MQBO must be in the character set and encoding of the local queue manager; these are given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE, respectively. However, if the application is running as an MQ client, the structure must be in the character set and encoding of the client.

Fields

The MQBO structure contains the following fields; the fields are described in **alphabetic order**:

Options (MQLONG)

Options that control the action of MQBEGIN.

The value must be:

MQBO_NONE

No options specified.

This is always an input field. The initial value of this field is MQBO_NONE.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQBO_STRUC_ID

Identifier for begin-options structure.

For the C programming language, the constant MQBO_STRUC_ID_ARRAY is also defined; this has the same value as MQBO_STRUC_ID, but is an array of characters instead of a string.

MQBO – Version field

This is always an input field. The initial value of this field is MQBO_STRUC_ID.

Version (MQLONG)

Structure version number.

The value must be:

MQBO_VERSION_1

Version number for begin-options structure.

The following constant specifies the version number of the current version:

MQBO_CURRENT_VERSION

Current version of begin-options structure.

This is always an input field. The initial value of this field is MQBO_VERSION_1.

Initial values and language declarations

Table 35. Initial values of fields in MQBO

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQBO_STRUC_ID	'B0bb'
<i>Version</i>	MQBO_VERSION_1	1
<i>Options</i>	MQBO_NONE	0

Notes:

1. The symbol 'b' represents a single blank character.
2. In the C programming language, the macro variable MQBO_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:
MQBO MyBO = {MQBO_DEFAULT};

C declaration

```
typedef struct tagMQBO MQBO;  
struct tagMQBO {  
    MQCHAR4  StrucId; /* Structure identifier */  
    MQLONG   Version; /* Structure version number */  
    MQLONG   Options; /* Options that control the action of MQBEGIN */  
};
```

COBOL declaration

```
** MQBO structure  
10 MQBO.  
** Structure identifier  
15 MQBO-STRUCID PIC X(4).  
** Structure version number  
15 MQBO-VERSION PIC S9(9) BINARY.  
** Options that control the action of MQBEGIN  
15 MQBO-OPTIONS PIC S9(9) BINARY.
```

PL/I declaration

```
dc1
1 MQBO based,
3 StrucId char(4),      /* Structure identifier */
3 Version fixed bin(31), /* Structure version number */
3 Options fixed bin(31); /* Options that control the action of
                        MQBEGIN */
```

Visual Basic declaration

```
Type MQBO
  StrucId As String*4 'Structure identifier'
  Version As Long     'Structure version number'
  Options As Long     'Options that control the action of MQBEGIN'
End Type
```

MQBO – Language declarations

Chapter 4. MQCIH – CICS bridge header

The following table summarizes the fields in the structure.

Table 36. Fields in MQCIH

Field	Description	Page
<i>StrucId</i>	Structure identifier	52
<i>Version</i>	Structure version number	54
<i>StrucLength</i>	Length of MQCIH structure	52
<i>Encoding</i>	Reserved	45
<i>CodedCharSetId</i>	Reserved	45
<i>Format</i>	MQ format name of data that follows MQCIH	47
<i>Flags</i>	Flags	46
<i>ReturnCode</i>	Return code from bridge	50
<i>CompCode</i>	MQ completion code or CICS EIBRESP	45
<i>Reason</i>	MQ reason or feedback code, or CICS EIBRESP2	49
<i>UOWControl</i>	Unit-of-work control	53
<i>GetWaitInterval</i>	Wait interval for MQGET call issued by bridge task	48
<i>LinkType</i>	Link type	48
<i>OutputDataLength</i>	Output COMMAREA data length	49
<i>FacilityKeepTime</i>	Bridge facility release time	46
<i>ADSDescriptor</i>	Send/receive ADS descriptor	43
<i>ConversationalTask</i>	Whether task can be conversational	45
<i>TaskEndStatus</i>	Status at end of task	52
<i>Facility</i>	Bridge facility token	46
<i>Function</i>	MQ call name or CICS EIBFN function	47
<i>AbendCode</i>	Abend code	43
<i>Authenticator</i>	Password or passticket	44
<i>Reserved1</i>	Reserved	50
<i>ReplyToFormat</i>	MQ format name of reply message	50
<i>RemoteSysId</i>	Reserved	49
<i>RemoteTransId</i>	Reserved	50
<i>TransactionId</i>	Transaction to attach	53
<i>FacilityLike</i>	Terminal emulated attributes	46
<i>AttentionId</i>	AID key	44
<i>StartCode</i>	Transaction start code	51
<i>CancelCode</i>	Abend transaction code	44
<i>NextTransactionId</i>	Next transaction to attach	49
<i>Reserved2</i>	Reserved	50
<i>Reserved3</i>	Reserved	50

MQCIH – CICS bridge header

Table 36. Fields in MQCIH (continued)

Field	Description	Page
Note: The remaining fields are not present if <i>Version</i> is less than MQCIH_VERSION_2.		
<i>CursorPosition</i>	Cursor position	45
<i>ErrorOffset</i>	Offset of error in message	45
<i>InputItem</i>	Reserved	48
<i>Reserved4</i>	Reserved	50

Overview

Availability: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Note: CICS connection is not available when using the reduced function form of WebSphere MQ for z/OS supplied with WebSphere Application Server.

Purpose: The MQCIH structure describes the information that can be present at the start of a message sent to the CICS bridge through WebSphere MQ for z/OS.

Format name: MQFMT_CICS.

Version: The current version of MQCIH is MQCIH_VERSION_2. Fields that exist only in the more-recent version of the structure are identified as such in the descriptions that follow.

The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQCIH, with the initial value of the *Version* field set to MQCIH_VERSION_2.

Character set and encoding: Special conditions apply to the character set and encoding used for the MQCIH structure and application message data:

- Applications that connect to the queue manager that owns the CICS bridge queue must provide an MQCIH structure that is in the character set and encoding of the queue manager. This is because data conversion of the MQCIH structure is not performed in this case.
- Applications that connect to other queue managers can provide an MQCIH structure that is in any of the supported character sets and encodings; conversion of the MQCIH is performed by the receiving message channel agent connected to the queue manager that owns the CICS bridge queue.

Note: There is one exception to this. If the queue manager that owns the CICS bridge queue is using CICS for distributed queuing, the MQCIH must be in the character set and encoding of the queue manager that owns the CICS bridge queue.

- The application message data following the MQCIH structure must be in the same character set and encoding as the MQCIH structure. The *CodedCharSetId* and *Encoding* fields in the MQCIH structure cannot be used to specify the character set and encoding of the application message data.

A data-conversion exit must be provided by the user to convert the application message data if the data is not one of the built-in formats supported by the queue manager.

MQCIH – CICS bridge header

Usage: If the values required by the application are the same as the initial values shown in Table 38 on page 54, and the bridge is running with AUTH=LOCAL or AUTH=IDENTIFY, the MQCIH structure can be omitted from the message. In all other cases, the structure must be present.

The bridge accepts either a version-1 or a version-2 MQCIH structure, but for 3270 transactions a version-2 structure must be used.

The application must ensure that fields documented as “request” fields have appropriate values in the message sent to the bridge; these fields are input to the bridge.

Fields documented as “response” fields are set by the CICS bridge in the reply message that the bridge sends to the application. Error information is returned in the *ReturnCode*, *Function*, *CompCode*, *Reason*, and *AbendCode* fields, but not all of them are set in all cases. Table 37 shows which fields are set for different values of *ReturnCode*.

Table 37. Contents of error information fields in MQCIH structure

<i>ReturnCode</i>	<i>Function</i>	<i>CompCode</i>	<i>Reason</i>	<i>AbendCode</i>
MQCRC_OK	–	–	–	–
MQCRC_BRIDGE_ERROR	–	–	MQFB_CICS_*	–
MQCRC_MQ_API_ERROR MQCRC_BRIDGE_TIMEOUT	MQ call name	MQ <i>CompCode</i>	MQ <i>Reason</i>	–
MQCRC_CICS_EXEC_ERROR MQCRC_SECURITY_ERROR MQCRC_PROGRAM_NOT_AVAILABLE MQCRC_TRANSID_NOT_AVAILABLE	CICS EIBFN	CICS EIBRESP	CICS EIBRESP2	–
MQCRC_BRIDGE_ABEND MQCRC_APPLICATION_ABEND	–	–	–	CICS ABCODE

Fields

The MQCIH structure contains the following fields; the fields are described in **alphabetic order**:

AbendCode (MQCHAR4)

Abend code.

The value returned in this field is significant only if the *ReturnCode* field has the value MQCRC_APPLICATION_ABEND or MQCRC_BRIDGE_ABEND. If it does, *AbendCode* contains the CICS ABCODE value.

This is a response field. The length of this field is given by MQ_ABEND_CODE_LENGTH. The initial value of this field is 4 blank characters.

ADSDescriptor (MQLONG)

Send/receive ADS descriptor.

This is an indicator specifying whether ADS descriptors should be sent on SEND and RECEIVE BMS requests. The following values are defined:

MQCADSD_NONE

Do not send or receive ADS descriptor.

MQCIH – ADSDescriptor field

MQCADSD_SEND

Send ADS descriptor.

MQCADSD_RECV

Receive ADS descriptor.

MQCADSD_MSGFORMAT

Use message format for the ADS descriptor.

This causes the ADS descriptor to be sent or received using the long form of the ADS descriptor. The long form has fields that are aligned on 4-byte boundaries.

The *ADSDescriptor* field should be set as follows:

- If ADS descriptors are *not* being used, set the field to MQCADSD_NONE.
- If ADS descriptors *are* being used, and with the *same* CCSID in each environment, set the field to the sum of MQCADSD_SEND and MQCADSD_RECV.
- If ADS descriptors *are* being used, but with *different* CCSIDs in each environment, set the field to the sum of MQCADSD_SEND, MQCADSD_RECV, and MQCADSD_MSGFORMAT.

This is a request field used only for 3270 transactions. The initial value of this field is MQCADSD_NONE.

AttentionId (MQCHAR4)

AID key.

This is the initial value of the AID key when the transaction is started. It is a 1-byte value, left justified.

This is a request field used only for 3270 transactions. The length of this field is given by MQ_ATTENTION_ID_LENGTH. The initial value of this field is 4 blanks.

Authenticator (MQCHAR8)

Password or passticket.

This is a password or passticket. If user-identifier authentication is active for the CICS bridge, *Authenticator* is used with the user identifier in the MQMD identity context to authenticate the sender of the message.

This is a request field. The length of this field is given by MQ_AUTHENTICATOR_LENGTH. The initial value of this field is 8 blanks.

CancelCode (MQCHAR4)

Abend transaction code.

This is the abend code to be used to terminate the transaction (normally a conversational transaction that is requesting more data). Otherwise this field is set to blanks.

This is a request field used only for 3270 transactions. The length of this field is given by MQ_CANCEL_CODE_LENGTH. The initial value of this field is 4 blanks.

CodedCharSetId (MQLONG)

Reserved.

This is a reserved field; its value is not significant. The initial value of this field is 0.

CompCode (MQLONG)

MQ completion code or CICS EIBRESP.

The value returned in this field is dependent on *ReturnCode*; see Table 37 on page 43.

This is a response field. The initial value of this field is MQCC_OK

ConversationalTask (MQLONG)

Whether task can be conversational.

This is an indicator specifying whether the task should be allowed to issue requests for more information, or should abend. The value must be one of the following:

MQCCT_YES

Task is conversational.

MQCCT_NO

Task is not conversational.

This is a request field used only for 3270 transactions. The initial value of this field is MQCCT_NO.

CursorPosition (MQLONG)

Cursor position.

This is the initial cursor position when the transaction is started. Subsequently, for conversational transactions, the cursor position is in the RECEIVE vector.

This is a request field used only for 3270 transactions. The initial value of this field is 0. This field is not present if *Version* is less than MQCIH_VERSION_2.

Encoding (MQLONG)

Reserved.

This is a reserved field; its value is not significant. The initial value of this field is 0.

ErrorOffset (MQLONG)

Offset of error in message.

This is the position of invalid data detected by the bridge exit. This field provides the offset from the start of the message to the location of the invalid data.

This is a response field used only for 3270 transactions. The initial value of this field is 0. This field is not present if *Version* is less than MQCIH_VERSION_2.

MQCIH – Facility field

Facility (MQBYTE8)

Bridge facility token.

This is an 8-byte bridge facility token. The purpose of a bridge facility token is to allow multiple transactions in a pseudoconversation to use the same bridge facility (virtual 3270 terminal). In the first, or only, message in a pseudoconversation, a value of MQCFAC_NONE should be set; this tells CICS to allocate a new bridge facility for this message. A bridge facility token is returned in response messages when a nonzero *FacilityKeepTime* is specified on the input message. Subsequent input messages can then use the same bridge facility token.

The following special value is defined:

MQCFAC_NONE

No BVT token specified.

For the C programming language, the constant MQCFAC_NONE_ARRAY is also defined; this has the same value as MQCFAC_NONE, but is an array of characters instead of a string.

This is both a request and a response field used only for 3270 transactions. The length of this field is given by MQ_FACILITY_LENGTH. The initial value of this field is MQCFAC_NONE.

FacilityKeepTime (MQLONG)

Bridge facility release time.

This is the length of time in seconds that the bridge facility will be kept after the user transaction has ended. For nonconversational transactions, the value should be zero.

This is a request field used only for 3270 transactions. The initial value of this field is 0.

FacilityLike (MQCHAR4)

Terminal emulated attributes.

This is the name of an installed terminal that is to be used as a model for the bridge facility. A value of blanks means that *FacilityLike* is taken from the bridge transaction profile definition, or a default value is used.

This is a request field used only for 3270 transactions. The length of this field is given by MQ_FACILITY_LIKE_LENGTH. The initial value of this field is 4 blanks.

Flags (MQLONG)

Flags.

The value must be:

MQCIH_NONE

No flags.

MQCIH_PASS_EXPIRATION

The reply message contains:

- The same expiry report options as the request message

- The remaining expiry time from the request message with no adjustment made for the bridge's processing time

If this value is not set, the expiry time is set to *unlimited*.

MQCIH_REPLY_WITHOUT_NULLS

The reply message length of a CICS DPL program request is adjusted to exclude trailing nulls (X'00') at the end of the COMMAREA returned by the DPL program. If this value is not set, the nulls might be significant, and the full COMMAREA is returned.

MQCIH_SYNC_ON_RETURN

The CICS link for DPL requests uses the SYNCONRETURN option. This causes CICS to take a syncpoint when the program completes if it is shipped to another CICS region. The bridge does not specify to which CICS region to ship the request; that is controlled by the CICS program definition or workload balancing facilities.

This is a request field. The initial value of this field is MQCIH_NONE.

Format (MQCHAR8)

MQ format name of data that follows MQCIH.

This specifies the MQ format name of the data that follows the MQCIH structure.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *Format* field in MQMD.

This format name is also used for the reply message, if the *ReplyToFormat* field has the value MQFMT_NONE.

- For DPL requests, *Format* must be the format name of the COMMAREA.
- For 3270 requests, *Format* must be CSQCBDCI, and *ReplyToFormat* must be CSQCBDCO.

The data-conversion exits for these formats must be installed on the queue manager where they are to run.

If the request message results in the generation of an error reply message, the error reply message has a format name of MQFMT_STRING.

This is a request field. The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

Function (MQCHAR4)

MQ call name or CICS EIBFN function.

The value returned in this field is dependent on *ReturnCode*; see Table 37 on page 43. The following values are possible when *Function* contains an MQ call name:

MQCFUNC_MQCONN
MQCONN call.

MQCFUNC_MQGET
MQGET call.

MQCIH – Function field

MQCFUNC_MQINQ

MQINQ call.

MQCFUNC_MQOPEN

MQOPEN call.

MQCFUNC_MQPUT

MQPUT call.

MQCFUNC_MQPUT1

MQPUT1 call.

MQCFUNC_NONE

No call.

In all cases, for the C programming language the constants `MQCFUNC_*_ARRAY` are also defined; these have the same values as the corresponding `MQCFUNC_*` constants, but are arrays of characters instead of strings.

This is a response field. The length of this field is given by `MQ_FUNCTION_LENGTH`. The initial value of this field is `MQCFUNC_NONE`.

GetWaitInterval (MQLONG)

Wait interval for `MQGET` call issued by bridge task.

This field is applicable only when *UOWControl* has the value `MQCUOWC_FIRST`. It allows the sending application to specify the approximate time in milliseconds that the `MQGET` calls issued by the bridge should wait for second and subsequent request messages for the unit of work started by this message. This overrides the default wait interval used by the bridge. The following special values may be used:

MQCGWI_DEFAULT

Default wait interval.

This causes the CICS bridge to wait for the period of time specified when the bridge was started.

MQWI_UNLIMITED

Unlimited wait interval.

This is a request field. The initial value of this field is `MQCGWI_DEFAULT`.

InputItem (MQLONG)

Reserved.

This is a reserved field. The value must be 0. This field is not present if *Version* is less than `MQCIH_VERSION_2`.

LinkType (MQLONG)

Link type.

This indicates the type of object that the bridge should try to link. The value must be one of the following:

MQCLT_PROGRAM

DPL program.

MQCLT_TRANSACTION

3270 transaction.

This is a request field. The initial value of this field is MQCLT_PROGRAM.

NextTransactionId (MQCHAR4)

Next transaction to attach.

This is the name of the next transaction returned by the user transaction (usually by EXEC CICS RETURN TRANSID). If there is no next transaction, this field is set to blanks.

This is a response field used only for 3270 transactions. The length of this field is given by MQ_TRANSACTION_ID_LENGTH. The initial value of this field is 4 blanks.

OutputDataLength (MQLONG)

Output COMMAREA data length.

This is the length of the user data to be returned to the client in a reply message. This length includes the 8-byte program name. The length of the COMMAREA passed to the linked program is the maximum of this field and the length of the user data in the request message, minus 8.

Note: The length of the user data in a message is the length of the message *excluding* the MQCIH structure.

If the length of the user data in the request message is smaller than *OutputDataLength*, the DATALENGTH option of the LINK command is used; this allows the LINK to be function-shipped efficiently to another CICS region.

The following special value can be used:

MQCODL_AS_INPUT

Output length is same as input length.

This value may be needed even if no reply is requested, in order to ensure that the COMMAREA passed to the linked program is of sufficient size.

This is a request field used only for DPL programs. The initial value of this field MQCODL_AS_INPUT.

Reason (MQLONG)

MQ reason or feedback code, or CICS EIBRESP2.

The value returned in this field is dependent on *ReturnCode*; see Table 37 on page 43.

This is a response field. The initial value of this field is MQRC_NONE.

RemoteSysId (MQCHAR4)

Reserved.

This is a reserved field. The value must be 4 blanks. The length of this field is given by MQ_REMOTE_SYS_ID_LENGTH.

MQCIH – RemoteTransId field

RemoteTransId (MQCHAR4)

Reserved.

This is a reserved field. The value must be 4 blanks. The length of this field is given by MQ_TRANSACTION_ID_LENGTH.

ReplyToFormat (MQCHAR8)

MQ format name of reply message.

This is the MQ format name of the reply message that will be sent in response to the current message. The rules for coding this are the same as those for the *Format* field in MQMD.

This is a request field used only for DPL programs. The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

Reserved1 (MQCHAR8)

Reserved.

This is a reserved field. The value must be 8 blanks.

Reserved2 (MQCHAR8)

Reserved.

This is a reserved field. The value must be 8 blanks.

Reserved3 (MQCHAR8)

Reserved.

This is a reserved field. The value must be 8 blanks.

Reserved4 (MQLONG)

Reserved.

This is a reserved field. The value must be 0. This field is not present if *Version* is less than MQCIH_VERSION_2.

ReturnCode (MQLONG)

Return code from bridge.

This is the return code from the CICS bridge describing the outcome of the processing performed by the bridge. The *Function*, *CompCode*, *Reason*, and *AbendCode* fields may contain additional information (see Table 37 on page 43). The value is one of the following:

MQCRC_APPLICATION_ABEND

(5, X'005') Application ended abnormally.

MQCRC_BRIDGE_ABEND

(4, X'004') CICS bridge ended abnormally.

MQCRC_BRIDGE_ERROR

(3, X'003') CICS bridge detected an error.

MQCRC_BRIDGE_TIMEOUT

(8, X'008') Second or later message within current unit of work not received within specified time.

MQCRC_CICS_EXEC_ERROR

(1, X'001') EXEC CICS statement detected an error.

MQCRC_MQ_API_ERROR

(2, X'002') MQ call detected an error.

MQCRC_OK

(0, X'000') No error.

MQCRC_PROGRAM_NOT_AVAILABLE

(7, X'007') Program not available.

MQCRC_SECURITY_ERROR

(6, X'006') Security error occurred.

MQCRC_TRANSID_NOT_AVAILABLE

(9, X'009') Transaction not available.

This is a response field. The initial value of this field is MQCRC_OK.

StartCode (MQCHAR4)

Transaction start code.

This is an indicator specifying whether the bridge emulates a terminal transaction or a STARTed transaction. The value must be one of the following:

MQCSC_START

Start.

MQCSC_STARTDATA

Start data.

MQCSC_TERMINPUT

Terminate input.

MQCSC_NONE

None.

In all cases, for the C programming language the constants MQCSC_*_ARRAY are also defined; these have the same values as the corresponding MQCSC_* constants, but are arrays of characters instead of strings.

In the response from the bridge, this field is set to the start code appropriate to the next transaction ID contained in the *NextTransactionId* field. The following start codes are possible in the response:

MQCSC_START

MQCSC_STARTDATA

MQCSC_TERMINPUT

For CICS Transaction Server Version 1.2, this field is a request field only; its value in the response is undefined.

For CICS Transaction Server Version 1.3 and subsequent releases, this is both a request and a response field.

MQCIH – StartCode field

This field is used only for 3270 transactions. The length of this field is given by `MQ_START_CODE_LENGTH`. The initial value of this field is `MQCSC_NONE`.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQCIH_STRUC_ID

Identifier for CICS information header structure.

For the C programming language, the constant `MQCIH_STRUC_ID_ARRAY` is also defined; this has the same value as `MQCIH_STRUC_ID`, but is an array of characters instead of a string.

This is a request field. The initial value of this field is `MQCIH_STRUC_ID`.

StrucLength (MQLONG)

Length of MQCIH structure.

The value must be one of the following:

MQCIH_LENGTH_1

Length of version-1 CICS information header structure.

MQCIH_LENGTH_2

Length of version-2 CICS information header structure.

The following constant specifies the length of the current version:

MQCIH_CURRENT_LENGTH

Length of current version of CICS information header structure.

This is a request field. The initial value of this field is `MQCIH_LENGTH_2`.

TaskEndStatus (MQLONG)

Status at end of task.

This field shows the status of the user transaction at end of task. One of the following values is returned:

MQCTES_NOSYNC

Not synchronized.

The user transaction has not yet completed and has not syncpointed. The *MsgType* field in MQMD is `MQMT_REQUEST` in this case.

MQCTES_COMMIT

Commit unit of work.

The user transaction has not yet completed, but has syncpointed the first unit of work. The *MsgType* field in MQMD is `MQMT_DATAGRAM` in this case.

MQCTES_BACKOUT

Back out unit of work.

The user transaction has not yet completed. The current unit of work will be backed out. The *MsgType* field in MQMD is `MQMT_DATAGRAM` in this case.

MQCTES_ENDTASK

End task.

The user transaction has ended (or abended). The *MsgType* field in MQMD is MQMT_REPLY in this case.

This is a response field used only for 3270 transactions. The initial value of this field is MQCTES_NOSYNC.

TransactionId (MQCHAR4)

Transaction to attach.

If *LinkType* has the value MQCLT_TRANSACTION, *TransactionId* is the transaction identifier of the user transaction to be run; a nonblank value must be specified in this case.

If *LinkType* has the value MQCLT_PROGRAM, *TransactionId* is the transaction code under which all programs within the unit of work are to be run. If the value specified is blank, the CICS DPL bridge default transaction code (CKBP) is used. If the value is nonblank, it must have been defined to CICS as a local TRANSACTION whose initial program is CSQCBP00. This field is applicable only when *UOWControl* has the value MQCUOWC_FIRST or MQCUOWC_ONLY.

This is a request field. The length of this field is given by MQ_TRANSACTION_ID_LENGTH. The initial value of this field is 4 blanks.

UOWControl (MQLONG)

Unit-of-work control.

This controls the unit-of-work processing performed by the CICS bridge. You can request the bridge to run a single transaction, or one or more programs within a unit of work. The field indicates whether the CICS bridge should start a unit of work, perform the requested function within the current unit of work, or end the unit of work by committing it or backing it out. Various combinations are supported, to optimize the data transmission flows.

The value must be one of the following:

MQCUOWC_ONLY

Start unit of work, perform function, then commit the unit of work (DPL and 3270).

MQCUOWC_CONTINUE

Additional data for the current unit of work (3270 only).

MQCUOWC_FIRST

Start unit of work and perform function (DPL only).

MQCUOWC_MIDDLE

Perform function within current unit of work (DPL only).

MQCUOWC_LAST

Perform function, then commit the unit of work (DPL only).

MQCUOWC_COMMIT

Commit the unit of work (DPL only).

MQCUOWC_BACKOUT

Back out the unit of work (DPL only).

MQCIH – Version field

This is a request field. The initial value of this field is MQCUOWC_ONLY.

Version (MQLONG)

Structure version number.

The value must be one of the following:

MQCIH_VERSION_1

Version-1 CICS information header structure.

MQCIH_VERSION_2

Version-2 CICS information header structure.

Fields that exist only in the more-recent version of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQCIH_CURRENT_VERSION

Current version of CICS information header structure.

This is a request field. The initial value of this field is MQCIH_VERSION_2.

Initial values and language declarations

Table 38. Initial values of fields in MQCIH

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQCIH_STRUC_ID	'CIHb'
<i>Version</i>	MQCIH_VERSION_2	2
<i>StrucLength</i>	MQCIH_LENGTH_2	180
<i>Encoding</i>	None	0
<i>CodedCharSetId</i>	None	0
<i>Format</i>	MQFMT_NONE	Blanks
<i>Flags</i>	MQCIH_NONE	0
<i>ReturnCode</i>	MQCRC_OK	0
<i>CompCode</i>	MQCC_OK	0
<i>Reason</i>	MQRC_NONE	0
<i>UOWControl</i>	MQCUOWC_ONLY	273
<i>GetWaitInterval</i>	MQCGWI_DEFAULT	-2
<i>LinkType</i>	MQCLT_PROGRAM	1
<i>OutputDataLength</i>	MQCODL_AS_INPUT	-1
<i>FacilityKeepTime</i>	None	0
<i>ADSDDescriptor</i>	MQCADSD_NONE	0
<i>ConversationalTask</i>	MQCCT_NO	0
<i>TaskEndStatus</i>	MQCTES_NOSYNC	0
<i>Facility</i>	MQCFAC_NONE	Nulls
<i>Function</i>	MQCFUNC_NONE	Blanks
<i>AbendCode</i>	None	Blanks
<i>Authenticator</i>	None	Blanks

Table 38. Initial values of fields in MQCIH (continued)

Field name	Name of constant	Value of constant
<i>Reserved1</i>	None	Blanks
<i>ReplyToFormat</i>	MQFMT_NONE	Blanks
<i>RemoteSysId</i>	None	Blanks
<i>RemoteTransId</i>	None	Blanks
<i>TransactionId</i>	None	Blanks
<i>FacilityLike</i>	None	Blanks
<i>AttentionId</i>	None	Blanks
<i>StartCode</i>	MQCSC_NONE	Blanks
<i>CancelCode</i>	None	Blanks
<i>NextTransactionId</i>	None	Blanks
<i>Reserved2</i>	None	Blanks
<i>Reserved3</i>	None	Blanks
<i>CursorPosition</i>	None	0
<i>ErrorOffset</i>	None	0
<i>InputItem</i>	None	0
<i>Reserved4</i>	None	0
<p>Notes:</p> <ol style="list-style-type: none"> 1. The symbol 'b' represents a single blank character. 2. In the C programming language, the macro variable MQCIH_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure: <pre>MQCIH MyCIH = {MQCIH_DEFAULT};</pre> 		

C declaration

```
typedef struct tagMQCIH MQCIH;
struct tagMQCIH {
    MQCHAR4  StrucId;           /* Structure identifier */
    MQLONG   Version;          /* Structure version number */
    MQLONG   StrucLength;      /* Length of MQCIH structure */
    MQLONG   Encoding;         /* Reserved */
    MQLONG   CodedCharSetId;   /* Reserved */
    MQCHAR8  Format;           /* MQ format name of data that follows
                               MQCIH */
    MQLONG   Flags;            /* Flags */
    MQLONG   ReturnCode;       /* Return code from bridge */
    MQLONG   CompCode;         /* MQ completion code or CICS EIBRESP */
    MQLONG   Reason;          /* MQ reason or feedback code, or CICS
                               EIBRESP2 */
    MQLONG   UOWControl;       /* Unit-of-work control */
    MQLONG   GetWaitInterval;  /* Wait interval for MQGET call issued
                               by bridge task */
    MQLONG   LinkType;         /* Link type */
    MQLONG   OutputDataLength; /* Output COMMAREA data length */
    MQLONG   FacilityKeepTime; /* Bridge facility release time */
    MQLONG   ADSDescriptor;    /* Send/receive ADS descriptor */
    MQLONG   ConversationalTask; /* Whether task can be conversational */
    MQLONG   TaskEndStatus;    /* Status at end of task */
    MQBYTE8  Facility;         /* Bridge facility token */
};
```

MQCIH – Language declarations

```
MQCHAR4 Function;           /* MQ call name or CICS EIBFN
                             function */
MQCHAR4 AbendCode;         /* Abend code */
MQCHAR8 Authenticator;     /* Password or passticket */
MQCHAR8 Reserved1;        /* Reserved */
MQCHAR8 ReplyToFormat;     /* MQ format name of reply message */
MQCHAR4 RemoteSysId;      /* Reserved */
MQCHAR4 RemoteTransId;    /* Reserved */
MQCHAR4 TransactionId;     /* Transaction to attach */
MQCHAR4 FacilityLike;     /* Terminal emulated attributes */
MQCHAR4 AttentionId;      /* AID key */
MQCHAR4 StartCode;        /* Transaction start code */
MQCHAR4 CancelCode;       /* Abend transaction code */
MQCHAR4 NextTransactionId; /* Next transaction to attach */
MQCHAR8 Reserved2;        /* Reserved */
MQCHAR8 Reserved3;        /* Reserved */
MQLONG CursorPosition;    /* Cursor position */
MQLONG ErrorOffset;       /* Offset of error in message */
MQLONG InputItem;         /* Reserved */
MQLONG Reserved4;        /* Reserved */
};
```

COBOL declaration

```
** MQCIH structure
10 MQCIH.
** Structure identifier
15 MQCIH-STRUCID          PIC X(4).
** Structure version number
15 MQCIH-VERSION         PIC S9(9) BINARY.
** Length of MQCIH structure
15 MQCIH-STRUCLength    PIC S9(9) BINARY.
** Reserved
15 MQCIH-ENCODING        PIC S9(9) BINARY.
** Reserved
15 MQCIH-CODEDCHARSETID  PIC S9(9) BINARY.
** MQ format name of data that follows MQCIH
15 MQCIH-FORMAT          PIC X(8).
** Flags
15 MQCIH-FLAGS           PIC S9(9) BINARY.
** Return code from bridge
15 MQCIH-RETURNCODE      PIC S9(9) BINARY.
** MQ completion code or CICS EIBRESP
15 MQCIH-COMPCODE        PIC S9(9) BINARY.
** MQ reason or feedback code, or CICS EIBRESP2
15 MQCIH-REASON          PIC S9(9) BINARY.
** Unit-of-work control
15 MQCIH-UOWCONTROL      PIC S9(9) BINARY.
** Wait interval for MQGET call issued by bridge task
15 MQCIH-GETWAITINTERVAL PIC S9(9) BINARY.
** Link type
15 MQCIH-LINKTYPE        PIC S9(9) BINARY.
** Output COMMAREA data length
15 MQCIH-OUTPUTDATALENGTH PIC S9(9) BINARY.
** Bridge facility release time
15 MQCIH-FACILITYKEEPTIME PIC S9(9) BINARY.
** Send/receive ADS descriptor
15 MQCIH-ADSDESCRIPTOR   PIC S9(9) BINARY.
** Whether task can be conversational
15 MQCIH-CONVERSATIONALTASK PIC S9(9) BINARY.
** Status at end of task
15 MQCIH-TASKENDSTATUS   PIC S9(9) BINARY.
** Bridge facility token
15 MQCIH-FACILITY        PIC X(8).
** MQ call name or CICS EIBFN function
15 MQCIH-FUNCTION        PIC X(4).
** Abend code
```

MQCIH – Language declarations

```
15 MQCIH-ABENDCODE          PIC X(4).
** Password or passticket
15 MQCIH-AUTHENTICATOR      PIC X(8).
** Reserved
15 MQCIH-RESERVED1          PIC X(8).
** MQ format name of reply message
15 MQCIH-REPLYTOFORMAT      PIC X(8).
** Reserved
15 MQCIH-REMOTESYSID        PIC X(4).
** Reserved
15 MQCIH-REMOETTRANSID      PIC X(4).
** Transaction to attach
15 MQCIH-TRANSACTIONID      PIC X(4).
** Terminal emulated attributes
15 MQCIH-FACILITYLIKE        PIC X(4).
** AID key
15 MQCIH-ATTENTIONID        PIC X(4).
** Transaction start code
15 MQCIH-STARTCODE          PIC X(4).
** Abend transaction code
15 MQCIH-CANCELCODE          PIC X(4).
** Next transaction to attach
15 MQCIH-NEXTTRANSACTIONID  PIC X(4).
** Reserved
15 MQCIH-RESERVED2          PIC X(8).
** Reserved
15 MQCIH-RESERVED3          PIC X(8).
** Cursor position
15 MQCIH-CURSORPOSITION      PIC S9(9) BINARY.
** Offset of error in message
15 MQCIH-ERROROFFSET        PIC S9(9) BINARY.
** Reserved
15 MQCIH-INPUTITEM          PIC S9(9) BINARY.
** Reserved
15 MQCIH-RESERVED4          PIC S9(9) BINARY.
```

PL/I declaration

```
dc1
1 MQCIH based,
3 StrucId          char(4),          /* Structure identifier */
3 Version          fixed bin(31),   /* Structure version number */
3 StrucLength      fixed bin(31),   /* Length of MQCIH structure */
3 Encoding         fixed bin(31),   /* Reserved */
3 CodedCharSetId   fixed bin(31),   /* Reserved */
3 Format            char(8),          /* MQ format name of data that
                                     follows MQCIH */
3 Flags            fixed bin(31),   /* Flags */
3 ReturnCode       fixed bin(31),   /* Return code from bridge */
3 CompCode         fixed bin(31),   /* MQ completion code or CICS
                                     EIBRESP */
3 Reason           fixed bin(31),   /* MQ reason or feedback code, or
                                     CICS EIBRESP2 */
3 UOWControl       fixed bin(31),   /* Unit-of-work control */
3 GetWaitInterval fixed bin(31),   /* Wait interval for MQGET call
                                     issued by bridge task */
3 LinkType         fixed bin(31),   /* Link type */
3 OutputDataLength fixed bin(31),   /* Output COMMAREA data length */
3 FacilityKeepTime fixed bin(31),   /* Bridge facility release time */
3 ADSDescriptor    fixed bin(31),   /* Send/receive ADS descriptor */
3 ConversationalTask fixed bin(31), /* Whether task can be
                                     conversational */
3 TaskEndStatus    fixed bin(31),   /* Status at end of task */
3 Facility         char(8),          /* Bridge facility token */
3 Function         char(4),          /* MQ call name or CICS EIBFN
                                     function */
3 AbendCode        char(4),          /* Abend code */
```

MQCIH – Language declarations

3 Authenticator	char(8),	/* Password or passticket */
3 Reserved1	char(8),	/* Reserved */
3 ReplyToFormat	char(8),	/* MQ format name of reply message */
3 RemoteSysId	char(4),	/* Reserved */
3 RemoteTransId	char(4),	/* Reserved */
3 TransactionId	char(4),	/* Transaction to attach */
3 FacilityLike	char(4),	/* Terminal emulated attributes */
3 AttentionId	char(4),	/* AID key */
3 StartCode	char(4),	/* Transaction start code */
3 CancelCode	char(4),	/* Abend transaction code */
3 NextTransactionId	char(4),	/* Next transaction to attach */
3 Reserved2	char(8),	/* Reserved */
3 Reserved3	char(8),	/* Reserved */
3 CursorPosition	fixed bin(31),	/* Cursor position */
3 ErrorOffset	fixed bin(31),	/* Offset of error in message */
3 InputItem	fixed bin(31),	/* Reserved */
3 Reserved4	fixed bin(31);	/* Reserved */

System/390 assembler declaration

MQCIH	DSECT	
MQCIH_STRUCID	DS CL4	Structure identifier
MQCIH_VERSION	DS F	Structure version number
MQCIH_STRUCLength	DS F	Length of MQCIH structure
MQCIH_ENCODING	DS F	Reserved
MQCIH_CODEDCHARSETID	DS F	Reserved
MQCIH_FORMAT	DS CL8	MQ format name of data that follows
*		MQCIH
MQCIH_FLAGS	DS F	Flags
MQCIH_RETURNCODE	DS F	Return code from bridge
MQCIH_COMPCODE	DS F	MQ completion code or CICS EIBRESP
MQCIH_REASON	DS F	MQ reason or feedback code, or CICS
*		EIBRESP2
MQCIH_UOWCONTROL	DS F	Unit-of-work control
MQCIH_GETWAITINTERVAL	DS F	Wait interval for MQGET call issued
*		by bridge task
MQCIH_LINKTYPE	DS F	Link type
MQCIH_OUTPUTDATALENGTH	DS F	Output COMMAREA data length
MQCIH_FACILITYKEEP TIME	DS F	Bridge facility release time
MQCIH_ADSDESCRIPTOR	DS F	Send/receive ADS descriptor
MQCIH_CONVERSATIONALTASK	DS F	Whether task can be conversational
MQCIH_TASKENDSTATUS	DS F	Status at end of task
MQCIH_FACILITY	DS XL8	Bridge facility token
MQCIH_FUNCTION	DS CL4	MQ call name or CICS EIBFN function
MQCIH_ABENDCODE	DS CL4	Abend code
MQCIH_AUTHENTICATOR	DS CL8	Password or passticket
MQCIH_RESERVED1	DS CL8	Reserved
MQCIH_REPLYTOFORMAT	DS CL8	MQ format name of reply message
MQCIH_REMOTESYSID	DS CL4	Reserved
MQCIH_REMOTETRANSID	DS CL4	Reserved
MQCIH_TRANSACTIONID	DS CL4	Transaction to attach
MQCIH_FACILITYLIKE	DS CL4	Terminal emulated attributes
MQCIH_ATTENTIONID	DS CL4	AID key
MQCIH_STARTCODE	DS CL4	Transaction start code
MQCIH_CANCELCODE	DS CL4	Abend transaction code
MQCIH_NEXTTRANSACTIONID	DS CL4	Next transaction to attach
MQCIH_RESERVED2	DS CL8	Reserved
MQCIH_RESERVED3	DS CL8	Reserved
MQCIH_CURSORPOSITION	DS F	Cursor position
MQCIH_ERROROFFSET	DS F	Offset of error in message
MQCIH_INPUTITEM	DS F	Reserved
MQCIH_RESERVED4	DS F	Reserved
*		
MQCIH_LENGTH	EQU	*-MQCIH
	ORG	MQCIH
MQCIH_AREA	DS	CL(MQCIH_LENGTH)

Visual Basic declaration

```

Type MQCIH
  StrucId          As String*4 'Structure identifier'
  Version          As Long     'Structure version number'
  StrucLength      As Long     'Length of MQCIH structure'
  Encoding         As Long     'Reserved'
  CodedCharSetId  As Long     'Reserved'
  Format           As String*8  'MQ format name of data that follows'
                    'MQCIH'
  Flags           As Long     'Flags'
  ReturnCode      As Long     'Return code from bridge'
  CompCode       As Long     'MQ completion code or CICS EIBRESP'
  Reason         As Long     'MQ reason or feedback code, or CICS'
                    'EIBRESP2'
  UOWControl     As Long     'Unit-of-work control'
  GetWaitInterval As Long     'Wait interval for MQGET call issued'
                    'by bridge task'
  LinkType       As Long     'Link type'
  OutputDataLength As Long     'Output COMMAREA data length'
  FacilityKeepTime As Long     'Bridge facility release time'
  ADSDescriptor  As Long     'Send/receive ADS descriptor'
  ConversationalTask As Long   'Whether task can be conversational'
  TaskEndStatus  As Long     'Status at end of task'
  Facility       As MQBYTE8  'Bridge facility token'
  Function       As String*4  'MQ call name or CICS EIBFN function'
  AbendCode     As String*4  'Abend code'
  Authenticator As String*8  'Password or passticket'
  Reserved1     As String*8  'Reserved'
  ReplyToFormat As String*8  'MQ format name of reply message'
  RemoteSysId   As String*4  'Reserved'
  RemoteTransId As String*4  'Reserved'
  TransactionId As String*4  'Transaction to attach'
  FacilityLike  As String*4  'Terminal emulated attributes'
  AttentionId  As String*4  'AID key'
  StartCode    As String*4  'Transaction start code'
  CancelCode   As String*4  'Abend transaction code'
  NextTransactionId As String*4 'Next transaction to attach'
  Reserved2    As String*8  'Reserved'
  Reserved3    As String*8  'Reserved'
  CursorPosition As Long     'Cursor position'
  ErrorOffset   As Long     'Offset of error in message'
  InputItem     As Long     'Reserved'
  Reserved4     As Long     'Reserved'
End Type

```

MQCIH – Language declarations

Chapter 5. MQCNO – Connect options

The following table summarizes the fields in the structure.

Table 39. Fields in MQCNO

Field	Description	Page
<i>StrucId</i>	Structure identifier	70
<i>Version</i>	Structure version number	70
<i>Options</i>	Options that control the action of MQCONN	65
Note: The remaining fields are ignored if <i>Version</i> is less than MQCNO_VERSION_2.		
<i>ClientConnOffset</i>	Offset of MQCD structure for client connection	62
<i>ClientConnPtr</i>	Address of MQCD structure for client connection	62
Note: The remaining fields are ignored if <i>Version</i> is less than MQCNO_VERSION_3.		
<i>ConnTag</i>	Queue-manager connection tag	64
Note: The remaining fields are ignored if <i>Version</i> is less than MQCNO_VERSION_4.		
<i>SSLConfigPtr</i>	Address of MQSCO structure for client connection	69
<i>SSLConfigOffset</i>	Offset of MQSCO structure for client connection	69

Overview

Availability:

- Versions 1 and 2: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems
- Version 3: AIX, HP-UX, z/OS, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems
- Version 4: AIX, HP-UX, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems

Purpose: The MQCNO structure allows the application to specify options relating to the connection to the local queue manager. The structure is an input/output parameter on the MQCONN call.

Version: The current version of MQCNO is MQCNO_VERSION_4, but this version is not supported in all environments (see above). Applications that are intended to be portable between several environments must ensure that the required version of MQCNO is supported in all of the environments concerned. Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions that follow.

The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQCNO that is supported by the environment, but with the initial value of the *Version* field set to MQCNO_VERSION_1. To use fields that are not present in the version-1 structure, the application must set the *Version* field to the version number of the version required.

MQCNO – Connect options

Character set and encoding: Data in MQCNO must be in the character set and encoding of the local queue manager; these are given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE, respectively. However, if the application is running as a WebSphere MQ client, the structure must be in the character set and encoding of the client.

Fields

The MQCNO structure contains the following fields; the fields are described in **alphabetic order**:

ClientConnOffset (MQLONG)

Offset of MQCD structure for client connection.

This is the offset in bytes of an MQCD channel definition structure from the start of the MQCNO structure. The offset can be positive or negative.

ClientConnOffset is used only when the application issuing the MQCONN call is running as a WebSphere MQ client. For information on how to use this field, see the description of the *ClientConnPtr* field.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQCNO_VERSION_2.

Note: This field is not applicable when using the reduced function form of WebSphere MQ for z/OS supplied with WebSphere Application Server.

ClientConnPtr (MQPTR)

Address of MQCD structure for client connection.

ClientConnOffset and *ClientConnPtr* are used only when the application issuing the MQCONN call is running as a WebSphere MQ client. By specifying one or other of these fields, the application can control the definition of the client connection channel by providing an MQCD channel definition structure that contains the values required.

If the application is running as a WebSphere MQ client but the application does not provide an MQCD structure, the MQSERVER environment variable is used to select the channel definition. If MQSERVER is not set, the client channel table is used.

If the application is not running as a WebSphere MQ client, *ClientConnOffset* and *ClientConnPtr* are ignored.

If the application provides an MQCD structure, the fields listed below must be set to the values required; other fields in MQCD are ignored. Character strings can be padded with blanks to the length of the field, or terminated by a null character. Refer to the *WebSphere MQ Intercommunication* book for more information about the fields in the MQCD structure.

Field in MQCD	Value
<i>ChannelName</i>	Channel name.
<i>Version</i>	Structure version number. Must not be less than MQCD_VERSION_7.
<i>TransportType</i>	Any supported transport type.
<i>ModeName</i>	LU 6.2 mode name.

Field in MQCD	Value
<i>TpName</i>	LU 6.2 transaction program name.
<i>SecurityExit</i>	Name of channel security exit.
<i>SendExit</i>	Name of channel send exit.
<i>ReceiveExit</i>	Name of channel receive exit.
<i>MaxMsgLength</i>	Maximum length in bytes of messages that can be sent over the client connection channel.
<i>SecurityUserData</i>	User data for security exit.
<i>SendUserData</i>	User data for send exit.
<i>ReceiveUserData</i>	User data for receive exit.
<i>UserIdentifier</i>	User identifier to be used to establish an LU 6.2 session.
<i>Password</i>	Password to be used to establish an LU 6.2 session.
<i>ConnectionName</i>	Connection name.
<i>HeartbeatInterval</i>	Time in seconds between heartbeat flows.
<i>StrucLength</i>	Length of the MQCD structure.
<i>ExitNameLength</i>	Length of exit names addressed by <i>SendExitPtr</i> and <i>ReceiveExitPtr</i> . Must be greater than zero if <i>SendExitPtr</i> or <i>ReceiveExitPtr</i> is set to a value that is not the null pointer.
<i>ExitDataLength</i>	Length of exit data addressed by <i>SendUserDataPtr</i> and <i>ReceiveUserDataPtr</i> . Must be greater than zero if <i>SendUserDataPtr</i> or <i>ReceiveUserDataPtr</i> is set to a value that is not the null pointer.
<i>SendExitsDefined</i>	Number of send exits addressed by <i>SendExitPtr</i> . If zero, <i>SendExit</i> and <i>SendUserData</i> provide the exit name and data. If greater than zero, <i>SendExitPtr</i> and <i>SendUserDataPtr</i> provide the exit names and data, and <i>SendExit</i> and <i>SendUserData</i> must be blank.
<i>ReceiveExitsDefined</i>	Number of receive exits addressed by <i>ReceiveExitPtr</i> . If zero, <i>ReceiveExit</i> and <i>ReceiveUserData</i> provide the exit name and data. If greater than zero, <i>ReceiveExitPtr</i> and <i>ReceiveUserDataPtr</i> provide the exit names and data, and <i>ReceiveExit</i> and <i>ReceiveUserData</i> must be blank.
<i>SendExitPtr</i>	Address of name of first send exit.
<i>SendUserDataPtr</i>	Address of data for first send exit.
<i>ReceiveExitPtr</i>	Address of name of first receive exit.
<i>ReceiveUserDataPtr</i>	Address of data for first receive exit.
<i>LongRemoteUserIdLength</i>	Length of long remote user identifier.
<i>LongRemoteUserIdPtr</i>	Address of long remote user identifier.
<i>RemoteSecurityId</i>	Remote security identifier.
<i>SSLCipherSpec</i>	SSL CipherSpec.
<i>SSLPeerNamePtr</i>	Address of the SSL peer name.
<i>SSLPeerNameLength</i>	Length of SSL peer name.
<i>KeepAliveInterval</i>	The value passed to the communications stack for keepalive timing for the channel

The channel definition structure can be provided in one of two ways:

- By using the offset field *ClientConnOffset*

In this case, the application should declare a compound structure containing an MQCNO followed by the channel definition structure MQCD, and set *ClientConnOffset* to the offset of the channel definition structure from the start of the MQCNO. Care must be taken to ensure that this offset is correct. *ClientConnPtr* must be set to the null pointer or null bytes.

MQCNO – ClientConnPtr field

Using *ClientConnOffset* is recommended for programming languages which do not support the pointer data type, or which implement the pointer data type in a fashion which is not portable to different environments (for example, the COBOL programming language).

For the Visual Basic programming language, a compound structure called MQCNOCD is provided in the header file CMQXB.BAS; this structure contains an MQCNO structure followed by an MQCD structure. MQCNOCD can be initialized by invoking the MQCNOCD_DEFAULTS subroutine. MQCNOCD is used with the MQCONNXAny variant of the MQCONNX call; see the description of the MQCONNX call for further details.

- By using the pointer field *ClientConnPtr*

In this case, the application can declare the channel definition structure separately from the MQCNO structure, and set *ClientConnPtr* to the address of the channel definition structure. *ClientConnOffset* must be set to zero.

Using *ClientConnPtr* is recommended for programming languages which support the pointer data type in a fashion which is portable to different environments (for example, the C programming language).

In the C programming language, the macro variable MQCD_CLIENT_CONN_DEFAULT can be used to provide initial values for the structure that are more suitable for use on the MQCONNX call than those provided by MQCD_DEFAULT.

Whichever technique is chosen, only one of *ClientConnOffset* and *ClientConnPtr* can be used; the call fails with reason code MQRC_CLIENT_CONN_ERROR if both are nonzero.

Once the MQCONNX call has completed, the MQCD structure is not referenced again.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQCNO_VERSION_2.

Notes:

1. On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.
2. This field is not applicable when using the reduced function form of WebSphere MQ for z/OS supplied with WebSphere Application Server.

ConnTag (MQBYTE128)

Queue-manager connection tag.

This is a tag that the queue manager associates with the resources that are affected by the application during this connection. Each application or application instance should use a different value for the tag, so that the queue manager can correctly serialize access to the affected resources. See the descriptions of the MQCNO_*_CONN_TAG_* options for further details. The tag ceases to be valid when the application terminates or issues the MQDISC call.

Note: Connection tag values beginning with MQ in upper, lower, or mixed case in either ASCII or EBCDIC are reserved for use by IBM products. Do not use connection tag values beginning with these letters.

The following special value can be used if no tag is required:

MQCT_NONE

No connection tag specified. On the reduced function form of WebSphere MQ for z/OS supplied with WebSphere Application Server, this is the only value available.

The value is binary zero for the length of the field.

For the C programming language, the constant MQCT_NONE_ARRAY is also defined; this has the same value as MQCT_NONE, but is an array of characters instead of a string.

This field is used only on z/OS. In other environments, the value MQCT_NONE should be specified.

This is an input field. The length of this field is given by MQ_CONN_TAG_LENGTH. The initial value of this field is MQCT_NONE. This field is ignored if *Version* is less than MQCNO_VERSION_3.

Options (MQLONG)

Options that control the action of MQCONN.

Binding options: The following options control the type of MQ binding that will be used; only one of these options can be specified:

MQCNO_STANDARD_BINDING

Standard binding.

This option causes the application and the local-queue-manager agent (the component that manages queuing operations) to run in separate units of execution (generally, in separate processes). This arrangement maintains the integrity of the queue manager, that is, it protects the queue manager from errant programs.

MQCNO_STANDARD_BINDING should be used in situations where the application may not have been fully tested, or may be unreliable or untrustworthy. MQCNO_STANDARD_BINDING is the default.

MQCNO_STANDARD_BINDING is defined to aid program documentation. It is not intended that this option be used with any other option controlling the type of binding used, but as its value is zero, such use cannot be detected.

This option is supported in all environments.

MQCNO_FASTPATH_BINDING

Fastpath binding.

This option causes the application and the local-queue-manager agent to be part of the same unit of execution. This is in contrast to the normal method of binding, where the application and the local-queue-manager agent run in separate units of execution.

MQCNO_FASTPATH_BINDING is ignored if the queue manager does not support this type of binding; processing continues as though the option had not been specified.

MQCNO_FASTPATH_BINDING may be of advantage in situations where the use of multiple processes is a significant performance overhead

MQCNO – Options field

compared to the overall resource used by the application. An application that uses the fastpath binding is known as a *trusted application*.

The following important points must be considered when deciding whether to use the fastpath binding:

- **Use of the MQCNO_FASTPATH_BINDING option compromises the integrity of the queue manager, because it permits a rogue application to alter or corrupt messages and other data areas belonging to the queue manager. It should therefore be considered for use *only* in situations where these issues have been fully evaluated.**
- The application must not use asynchronous signals or timer interrupts (such as sigkill) with MQCNO_FASTPATH_BINDING. There are also restrictions on the use of shared memory segments. Refer to the *WebSphere MQ Application Programming Guide* for more information.
- The application must use the MQDISC call to disconnect from the queue manager.
- The application must finish before ending the queue manager with the endmqm command.

The following points apply to the use of MQCNO_FASTPATH_BINDING in the environments indicated:

- On OS/400, the job must run under a user profile that belongs to the QMQMADM group. Also, the program must not terminate abnormally, otherwise unpredictable results may occur.
- On UNIX systems, the mqm user identifier and the mqm group identifier must be the effective user identifier and group identifier respectively. The application can be made to run this way by configuring the program so that it is owned by the mqm user identifier and mqm group identifier, and then setting the setuid and setgid permission bits on the program.

The real user ID is still used by the WebSphere MQ Object Authority Manager (OAM) for authority checking.

- On Compaq NonStop Kernel, you can use fastpath binding only in a process that has a single connection to a queue manager. Additionally, the application must run under the user identifier that is part of the mqm group that created the queue manager.

You can use the Guardian parameter MQCONNECTTYPE in association with the bind type specified by the *Options* field, to control the type of binding used. If you use this parameter, it must have the value **FASTPATH** or **STANDARD**; if it has some other value, it is ignored. The value of the parameter is case sensitive.

- On Windows, the program must be a member of the mqm group.

For more information about the implications of using trusted applications, see the *WebSphere MQ Application Programming Guide*.

This option is supported in the following environments: AIX, HP-UX, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows. On z/OS the option is accepted but ignored.

On AIX, HP-UX, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, Solaris, Linux, and Windows, the environment variable MQ_CONNECT_TYPE can be used in association with the bind type specified by the *Options* field, to control the type of binding used. If this environment variable is specified, it should have the

value FASTPATH or STANDARD; if it has some other value, it is ignored. The value of the environment variable is case sensitive.

The environment variable and *Options* field interact as follows:

- If the environment variable is not specified, or has a value which is not supported, use of the fastpath binding is determined solely by the *Options* field.
- If the environment variable is specified and has a supported value, the fastpath binding is used only if *both* the environment variable and *Options* field specify the fastpath binding.

Connection-tag options: The following options control the use of the connection tag *ConnTag*. Only one of these options can be specified.

- These options are supported only on z/OS. However, they are not applicable when using WebSphere Application Server embedded messaging using reduced function WebSphere MQ on z/OS.

MQCNO_SERIALIZE_CONN_TAG_Q_MGR

Connection tag use is serialized within the queue manager.

This option requests exclusive use of the connection tag within the local queue manager. If the connection tag is already in use in the local queue manager, the MQCONNX call fails with reason code MQRC_CONN_TAG_IN_USE. The outcome of the call is not affected by use of the connection tag elsewhere in the queue-sharing group to which the local queue manager belongs.

MQCNO_SERIALIZE_CONN_TAG_QSG

Connection tag use is serialized within the queue-sharing group.

This option requests exclusive use of the connection tag within the queue-sharing group to which the local queue manager belongs. If the connection tag is already in use in the queue-sharing group, the MQCONNX call fails with reason code MQRC_CONN_TAG_IN_USE.

MQCNO_RESTRICT_CONN_TAG_Q_MGR

Connection tag use is restricted within the queue manager.

This option requests shared use of the connection tag within the local queue manager. If the connection tag is already in use in the local queue manager, the MQCONNX call can succeed provided that the requesting application is running in the same processing scope as the existing user of the tag. If this condition is not satisfied, the MQCONNX call fails with reason code MQRC_CONN_TAG_IN_USE. The outcome of the call is not affected by use of the connection tag elsewhere in the queue-sharing group to which the local queue manager belongs.

- On z/OS, applications must run within the same MVS address space in order to share the connection tag.

MQCNO_RESTRICT_CONN_TAG_QSG

Connection tag use is restricted within the queue-sharing group.

This option requests shared use of the connection tag within the queue-sharing group to which the local queue manager belongs. If the connection tag is already in use in the queue-sharing group, the MQCONNX call can succeed provided that:

- The requesting application is running in the same processing scope as the existing user of the tag.

MQCNO – Options field

- The requesting application is connected to the same queue manager as the existing user of the tag.

If these conditions are not satisfied, the MQCONNX call fails with reason code MQRC_CONN_TAG_IN_USE.

- On z/OS, applications must run within the same MVS address space in order to share the connection tag.

If none of these options is specified, *ConnTag* is not used. These options are not valid if *Version* is less than MQCNO_VERSION_3.

Handle-sharing options:The following options control the sharing of handles between different threads (units of parallel processing) within the same process. Only one of these options can be specified.

- These options are supported in the following environments: AIX, HP-UX, OS/400, Solaris, Linux, Windows.

MQCNO_HANDLE_SHARE_NONE

No handle sharing between threads.

This option indicates that connection and object handles can be used only by the thread that caused the handle to be allocated (that is, the thread that issued the MQCONN, MQCONNX, or MQOPEN call). The handles cannot be used by other threads belonging to the same process.

MQCNO_HANDLE_SHARE_BLOCK

Serial handle sharing between threads, with call blocking.

This option indicates that connection and object handles allocated by one thread of a process can be used by other threads belonging to the same process. However, only one thread at a time can use any particular handle, that is, only serial use of a handle is permitted. If a thread tries to use a handle that is already in use by another thread, the call blocks (waits) until the handle becomes available.

MQCNO_HANDLE_SHARE_NO_BLOCK

Serial handle sharing between threads, without call blocking.

This is the same as MQCNO_HANDLE_SHARE_BLOCK, except that if the handle is in use by another thread, the call completes immediately with MQCC_FAILED and MQRC_CALL_IN_PROGRESS instead of blocking until the handle becomes available.

A thread can have zero or one nonshared handle:

- Each MQCONN or MQCONNX call that specifies MQCNO_HANDLE_SHARE_NONE returns a new nonshared handle on the first call, and the same nonshared handle on the second and later calls (assuming no intervening MQDISC call). The reason code is MQRC_ALREADY_CONNECTED for the second and later calls.
- Each MQCONNX call that specifies MQCNO_HANDLE_SHARE_BLOCK or MQCNO_HANDLE_SHARE_NO_BLOCK returns a new shared handle on each call.

Object handles inherit the same shareability as the connection handle specified on the MQOPEN call that created the object handle. Also, units of work inherit the same shareability as the connection handle used to start the unit of work; if the unit of work is started in one thread using a shared handle, the unit of work can be updated in another thread using the same handle.

If no handle-sharing option is specified, the default is determined by the environment:

- In the Microsoft Transaction Server (MTS) environment, the default is the same as MQCNO_HANDLE_SHARE_BLOCK.
- In other environments, the default is the same as MQCNO_HANDLE_SHARE_NONE.

Default option: If none of the options described above is required, the following option can be used:

MQCNO_NONE

No options specified.

MQCNO_NONE is defined to aid program documentation. It is not intended that this option be used with any other MQCNO_* option, but as its value is zero, such use cannot be detected.

This is always an input field. The initial value of this field is MQCNO_NONE.

SSLConfigOffset (MQLONG)

Offset of MQSCO structure for client connection.

This is the offset in bytes of an MQSCO structure from the start of the MQCNO structure. The offset can be positive or negative.

SSLConfigOffset is used only when the application issuing the MQCONN call is running as a WebSphere MQ client. For information on how to use this field, see the description of the *SSLConfigPtr* field.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQCNO_VERSION_4.

SSLConfigPtr (PMQSCO)

Address of MQSCO structure for client connection.

SSLConfigPtr and *SSLConfigOffset* are used only when the application issuing the MQCONN call is running as a WebSphere MQ client and the channel protocol is TCP/IP. If the application is not running as a WebSphere MQ client, or the channel protocol is not TCP/IP, *SSLConfigPtr* and *SSLConfigOffset* are ignored.

By specifying *SSLConfigPtr* or *SSLConfigOffset*, plus either *ClientConnPtr* or *ClientConnOffset*, the application can control the use of SSL for the client connection. When the SSL information is specified in this way, the environment variables MQSSLKEYR and MQSSLCRYP are ignored; any SSL-related information in the client channel definition table is also ignored.

The SSL information can be specified only on:

- The first MQCONN call of the client process, or
- A subsequent MQCONN call when all previous connections to the queue manager (using any protocol) have been concluded using MQDISC.

These are the only states in which the process-wide SSL environment can be initialized. If an MQCONN call is issued specifying SSL information when the SSL environment already exists, the SSL information on the call is ignored and the

MQCNO – SSLConfigPtr field

connection is made using the existing SSL environment; the call returns completion code MQCC_WARNING and reason code MQRC_SSL_ALREADY_INITIALIZED in this case.

The MQSCO structure can be provided in the same way as the MQCD structure, either by specifying an address in *SSLConfigPtr*, or by specifying an offset in *SSLConfigOffset*; see the description of *ClientConnPtr* for details of how to do this. However, no more than one of *SSLConfigPtr* and *SSLConfigOffset* can be used; the call fails with reason code MQRC_SSL_CONFIG_ERROR. if both are nonzero.

Once the MQCONN call has completed, the MQSCO structure is not referenced again.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQCNO_VERSION_4.

Note: On platforms where the programming language does not support the pointer datatype, this field is declared as a byte string of the appropriate length.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQCNO_STRUC_ID

Identifier for connect-options structure.

For the C programming language, the constant MQCNO_STRUC_ID_ARRAY is also defined; this has the same value as MQCNO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQCNO_STRUC_ID.

Version (MQLONG)

Structure version number.

The value must be one of the following:

MQCNO_VERSION_1

Version-1 connect-options structure.

This version is supported in all environments.

MQCNO_VERSION_2

Version-2 connect-options structure.

This version is supported in all environments.

MQCNO_VERSION_3

Version-3 connect-options structure.

This version is supported in the following environments: AIX, HP-UX, z/OS, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

MQCNO_VERSION_4

Version-4 connect-options structure.

This version is supported in the following environments: AIX, HP-UX, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQCNO_CURRENT_VERSION

Current version of connect-options structure.

This is always an input field. The initial value of this field is MQCNO_VERSION_1.

Initial values and language declarations

Table 40. Initial values of fields in MQCNO

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQCNO_STRUC_ID	'CNOb'
<i>Version</i>	MQCNO_VERSION_1	1
<i>Options</i>	MQCNO_NONE	0
<i>ClientConnOffset</i>	None	0
<i>ClientConnPtr</i>	None	Null pointer or null bytes
<i>ConnTag</i>	MQCT_NONE	Nulls
<i>SSLConfigPtr</i>	None	Null pointer or null bytes
<i>SSLConfigOffset</i>	None	0

Notes:

1. The symbol 'b' represents a single blank character.
2. In the C programming language, the macro variable MQCNO_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQCNO MyCNO = {MQCNO_DEFAULT};
```

C declaration

```
typedef struct tagMQCNO MQCNO;
struct tagMQCNO {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQLONG    Options;          /* Options that control the action of
                                MQCONNX */
    MQLONG    ClientConnOffset; /* Offset of MQCD structure for client
                                connection */
    MQPTR     ClientConnPtr;    /* Address of MQCD structure for client
                                connection */
    MQBYTE128 ConnTag;          /* Queue-manager connection tag */
    PMQSCO    SSLConfigPtr;     /* Address of MQSCO structure for client
```

MQCNO – Language declarations

```
|
|
|           MQLONG      SSLConfigOffset; /* Offset of MQSCO structure for client
|                                           connection */
|
|};
```

COBOL declaration

```
** MQCNO structure
10 MQCNO.
** Structure identifier
15 MQCNO-STRUCID      PIC X(4).
** Structure version number
15 MQCNO-VERSION     PIC S9(9) BINARY.
** Options that control the action of MQCONN
15 MQCNO-OPTIONS     PIC S9(9) BINARY.
** Offset of MQCD structure for client connection
15 MQCNO-CLIENTCONNOFFSET PIC S9(9) BINARY.
** Address of MQCD structure for client connection
15 MQCNO-CLIENTCONNPTR  POINTER.
** Queue-manager connection tag
15 MQCNO-CONNTAG      PIC X(128).
** Address of MQSCO structure for client connection
15 MQCNO-SSLCONFIGPTR  POINTER.
** Offset of MQSCO structure for client connection
15 MQCNO-SSLCONFIGOFFSET PIC S9(9) BINARY.
```

PL/I declaration

```
dc1
1 MQCNO based,
3 StrucId      char(4), /* Structure identifier */
3 Version      fixed bin(31), /* Structure version number */
3 Options      fixed bin(31), /* Options that control the action
                             of MQCONN */
3 ClientConnOffset fixed bin(31), /* Offset of MQCD structure for
                             client connection */
3 ClientConnPtr  pointer, /* Address of MQCD structure for
                             client connection */
3 ConnTag        char(128), /* Queue-manager connection tag */
3 SSLConfigPtr   pointer, /* Address of MQSCO structure for
                             client connection */
3 SSLConfigOffset fixed bin(31); /* Offset of MQSCO structure for
                             client connection */
```

System/390 assembler declaration

```
MQCNO          DSECT
MQCNO_STRUCID  DS   CL4  Structure identifier
MQCNO_VERSION  DS   F    Structure version number
MQCNO_OPTIONS  DS   F    Options that control the action of
*                MQCONN
MQCNO_CLIENTCONNOFFSET DS F    Offset of MQCD structure for client
*                connection
MQCNO_CLIENTCONNPTR  DS   F    Address of MQCD structure for client
*                connection
MQCNO_CONNTAG   DS   XL128 Queue-manager connection tag
*
MQCNO_LENGTH    EQU  *-MQCNO
                ORG  MQCNO
MQCNO_AREA      DS   CL(MQCNO_LENGTH)
```

Visual Basic declaration

```
Type MQCNO
  StrucId      As String*4 'Structure identifier'
  Version      As Long     'Structure version number'
```

MQCNO – Language declarations

```
Options          As Long      'Options that control the action of'
                  'MQCONNX'
ClientConnOffset As Long      'Offset of MQCD structure for client'
                  'connection'
ClientConnPtr    As MQPTR     'Address of MQCD structure for client'
                  'connection'
ConnTag          As MQBYTE128 'Queue-manager connection tag'
SSLConfigPtr     As MQPTR     'Address of MQSCO structure for client'
                  'connection'
SSLConfigOffset  As Long      'Offset of MQSCO structure for client'
                  'connection'
End Type
```

MQCNO – Language declarations

Chapter 6. MQDH – Distribution header

The following table summarizes the fields in the structure.

Table 41. Fields in MQDH

Field	Description	Page
<i>StrucId</i>	Structure identifier	79
<i>Version</i>	Structure version number	80
<i>StrucLength</i>	Length of MQDH structure plus following records	79
<i>Encoding</i>	Numeric encoding of data that follows array of MQPMR records	77
<i>CodedCharSetId</i>	Character set identifier of data that follows array of MQPMR records	76
<i>Format</i>	Format name of data that follows array of MQPMR records	78
<i>Flags</i>	General flags	77
<i>PutMsgRecFields</i>	Flags indicating which MQPMR fields are present	78
<i>RecsPresent</i>	Number of object records present	79
<i>ObjectRecOffset</i>	Offset of first object record from start of MQDH	78
<i>PutMsgRecOffset</i>	Offset of first put-message record from start of MQDH	79

Overview

Availability: AIX, HP-UX, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Purpose: The MQDH structure describes the additional data that is present in a message when that message is a distribution-list message stored on a transmission queue. A distribution-list message is a message that is sent to multiple destination queues. The additional data consists of the MQDH structure followed by an array of MQOR records and an array of MQPMR records.

This structure is for use by specialized applications that put messages directly on transmission queues, or which remove messages from transmission queues (for example: message channel agents).

This structure should *not* be used by normal applications which simply want to put messages to distribution lists. Those applications should use the MQOD structure to define the destinations in the distribution list, and the MQPMO structure to specify message properties or receive information about the messages sent to the individual destinations.

Format name: MQFMT_DIST_HEADER.

MQDH – Distribution header

Character set and encoding: Data in MQDH must be in the character set and encoding of the local queue manager; these are given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE for the C programming language, respectively.

The character set and encoding of the MQDH must be set into the *CodedCharSetId* and *Encoding* fields in:

- The MQMD (if the MQDH structure is at the start of the message data), or
- The header structure that precedes the MQDH structure (all other cases).

Usage: When an application puts a message to a distribution list, and some or all of the destinations are remote, the queue manager prefixes the application message data with the MQXQH and MQDH structures, and places the message on the relevant transmission queue. The data therefore occurs in the following sequence when the message is on a transmission queue:

- MQXQH structure
- MQDH structure plus arrays of MQOR and MQPMR records
- Application message data

Depending on the destinations, more than one such message may be generated by the queue manager, and placed on different transmission queues. In this case, the MQDH structures in those messages identify different subsets of the destinations defined by the distribution list opened by the application.

An application that puts a distribution-list message directly on a transmission queue must conform to the sequence described above, and must ensure that the MQDH structure is correct. If the MQDH structure is not valid, the queue manager may choose to fail the MQPUT or MQPUT1 call with reason code MQRC_DH_ERROR.

Messages can be stored on a queue in distribution-list form only if the queue is defined as being able to support distribution list messages (see the *DistLists* queue attribute described in Chapter 40, “Attributes for queues”, on page 457). If an application puts a distribution-list message directly on a queue that does not support distribution lists, the queue manager splits the distribution list message into individual messages, and places those on the queue instead.

Fields

The MQDH structure contains the following fields; the fields are described in **alphabetic order**:

CodedCharSetId (MQLONG)

Character set identifier of data that follows the MQOR and MQPMR records.

This specifies the character set identifier of the data that follows the arrays of MQOR and MQPMR records; it does not apply to character data in the MQDH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

MQCCSI_INHERIT

Inherit character-set identifier of this structure.

MQDH – CodedCharSetId field

Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

MQCCSI_INHERIT cannot be used if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

This value is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG)

Numeric encoding of data that follows the MQOR and MQPMR records.

This specifies the numeric encoding of the data that follows the arrays of MQOR and MQPMR records; it does not apply to numeric data in the MQDH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is 0.

Flags (MQLONG)

General flags.

The following flag can be specified:

MQDHF_NEW_MSG_IDS

Generate new message identifiers.

This flag indicates that a new message identifier is to be generated for each destination in the distribution list. This can be set only when there are no put-message records present, or when the records are present but they do not contain the *MsgId* field.

Using this flag defers generation of the message identifiers until the last possible moment, namely the moment when the distribution-list message is finally split into individual messages. This minimizes the amount of control information that must flow with the distribution-list message.

When an application puts a message to a distribution list, the queue manager sets MQDHF_NEW_MSG_IDS in the MQDH it generates when both of the following are true:

- There are no put-message records provided by the application, or the records provided do not contain the *MsgId* field.
- The *MsgId* field in MQMD is MQMI_NONE, or the *Options* field in MQPMO includes MQPMO_NEW_MSG_ID

If no flags are needed, the following can be specified:

MQDHF_NONE

No flags.

MQDH – Flags field

This constant indicates that no flags have been specified. MQDHF_NONE is defined to aid program documentation. It is not intended that this constant be used with any other, but as its value is zero, such use cannot be detected.

The initial value of this field is MQDHF_NONE.

Format (MQCHAR8)

Format name of data that follows the MQOR and MQPMR records.

This specifies the format name of the data that follows the arrays of MQOD and MQPMR records (whichever occurs last).

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *Format* field in MQMD.

The initial value of this field is MQFMT_NONE.

ObjectRecOffset (MQLONG)

Offset of first MQOR record from start of MQDH.

This field gives the offset in bytes of the first record in the array of MQOR object records containing the names of the destination queues. There are *RecsPresent* records in this array. These records (plus any bytes skipped between the first object record and the previous field) are included in the length given by the *StrucLength* field.

A distribution list must always contain at least one destination, so *ObjectRecOffset* must always be greater than zero.

The initial value of this field is 0.

PutMsgRecFields (MQLONG)

Flags indicating which MQPMR fields are present.

Zero or more of the following flags can be specified:

MQPMRF_MSG_ID

Message-identifier field is present.

MQPMRF_CORREL_ID

Correlation-identifier field is present.

MQPMRF_GROUP_ID

Group-identifier field is present.

MQPMRF_FEEDBACK

Feedback field is present.

MQPMRF_ACCOUNTING_TOKEN

Accounting-token field is present.

If no MQPMR fields are present, the following can be specified:

MQPMRF_NONE

No put-message record fields are present.

MQDH – PutMsgRecFields field

MQPMRF_NONE is defined to aid program documentation. It is not intended that this constant be used with any other, but as its value is zero, such use cannot be detected.

The initial value of this field is MQPMRF_NONE.

PutMsgRecOffset (MQLONG)

Offset of first MQPMR record from start of MQDH.

This field gives the offset in bytes of the first record in the array of MQPMR put message records containing the message properties. If present, there are *RecsPresent* records in this array. These records (plus any bytes skipped between the first put message record and the previous field) are included in the length given by the *StrucLength* field.

Put message records are optional; if no records are provided, *PutMsgRecOffset* is zero, and *PutMsgRecFields* has the value MQPMRF_NONE.

The initial value of this field is 0.

RecsPresent (MQLONG)

Number of MQOR records present.

This defines the number of destinations. A distribution list must always contain at least one destination, so *RecsPresent* must always be greater than zero.

The initial value of this field is 0.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQDH_STRUC_ID

Identifier for distribution header structure.

For the C programming language, the constant MQDH_STRUC_ID_ARRAY is also defined; this has the same value as MQDH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQDH_STRUC_ID.

StrucLength (MQLONG)

Length of MQDH structure plus following MQOR and MQPMR records.

This is the number of bytes from the start of the MQDH structure to the start of the message data following the arrays of MQOR and MQPMR records. The data occurs in the following sequence:

- MQDH structure
- Array of MQOR records
- Array of MQPMR records
- Message data

The arrays of MQOR and MQPMR records are addressed by offsets contained within the MQDH structure. If these offsets result in unused bytes between one or

MQDH – StrucLength field

more of the MQDH structure, the arrays of records, and the message data, those unused bytes must be included in the value of *StrucLength*, but the content of those bytes is not preserved by the queue manager. It is valid for the array of MQPMR records to precede the array of MQOR records.

The initial value of this field is 0.

Version (MQLONG)

Structure version number.

The value must be:

MQDH_VERSION_1

Version number for distribution header structure.

The following constant specifies the version number of the current version:

MQDH_CURRENT_VERSION

Current version of distribution header structure.

The initial value of this field is MQDH_VERSION_1.

Initial values and language declarations

Table 42. Initial values of fields in MQDH

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQDH_STRUC_ID	'DHbb'
<i>Version</i>	MQDH_VERSION_1	1
<i>StrucLength</i>	None	0
<i>Encoding</i>	None	0
<i>CodedCharSetId</i>	MQCCSI_UNDEFINED	0
<i>Format</i>	MQFMT_NONE	Blanks
<i>Flags</i>	MQDHF_NONE	0
<i>PutMsgRecFields</i>	MQPMRF_NONE	0
<i>RecsPresent</i>	None	0
<i>ObjectRecOffset</i>	None	0
<i>PutMsgRecOffset</i>	None	0

Notes:

1. The symbol 'b' represents a single blank character.
2. In the C programming language, the macro variable MQDH_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQDH MyDH = {MQDH_DEFAULT};
```

C declaration

```
typedef struct tagMQDH MQDH;  
struct tagMQDH {  
    MQCHAR4  StrucId;          /* Structure identifier */  
    MQLONG   Version;         /* Structure version number */  
};
```

MQDH – Language declarations

```

MQLONG  StrucLength;      /* Length of MQDH structure plus following
                          MQOR and MQPMPR records */
MQLONG  Encoding;        /* Numeric encoding of data that follows
                          the MQOR and MQPMPR records */
MQLONG  CodedCharSetId;  /* Character set identifier of data that
                          follows the MQOR and MQPMPR records */
MQCHAR8  Format;          /* Format name of data that follows the
                          MQOR and MQPMPR records */
MQLONG  Flags;           /* General flags */
MQLONG  PutMsgRecFields; /* Flags indicating which MQPMPR fields are
                          present */
MQLONG  RecsPresent;     /* Number of MQOR records present */
MQLONG  ObjectRecOffset; /* Offset of first MQOR record from start
                          of MQDH */
MQLONG  PutMsgRecOffset; /* Offset of first MQPMPR record from start
                          of MQDH */
};
```

COBOL declaration

```

**  MQDH structure
   10 MQDH.
**  Structure identifier
   15 MQDH-STRUCID      PIC X(4).
**  Structure version number
   15 MQDH-VERSION     PIC S9(9) BINARY.
**  Length of MQDH structure plus following MQOR and MQPMPR records
   15 MQDH-STRUCLENGTH PIC S9(9) BINARY.
**  Numeric encoding of data that follows the MQOR and MQPMPR records
   15 MQDH-ENCODING    PIC S9(9) BINARY.
**  Character set identifier of data that follows the MQOR and MQPMPR
**  records
   15 MQDH-CODEDCHARSETID PIC S9(9) BINARY.
**  Format name of data that follows the MQOR and MQPMPR records
   15 MQDH-FORMAT      PIC X(8).
**  General flags
   15 MQDH-FLAGS       PIC S9(9) BINARY.
**  Flags indicating which MQPMPR fields are present
   15 MQDH-PUTMSGRECFIELDS PIC S9(9) BINARY.
**  Number of MQOR records present
   15 MQDH-RECSPRESENT PIC S9(9) BINARY.
**  Offset of first MQOR record from start of MQDH
   15 MQDH-OBJECTRECOFFSET PIC S9(9) BINARY.
**  Offset of first MQPMPR record from start of MQDH
   15 MQDH-PUTMSGRECOFFSET PIC S9(9) BINARY.
```

PL/I declaration

```

dc1
  1 MQDH based,
    3 StrucId      char(4),      /* Structure identifier */
    3 Version      fixed bin(31), /* Structure version number */
    3 StrucLength  fixed bin(31), /* Length of MQDH structure plus
                                following MQOR and MQPMPR
                                records */
    3 Encoding     fixed bin(31), /* Numeric encoding of data that
                                follows the MQOR and MQPMPR
                                records */
    3 CodedCharSetId fixed bin(31), /* Character set identifier of data
                                that follows the MQOR and MQPMPR
                                records */
    3 Format        char(8),      /* Format name of data that follows
                                the MQOR and MQPMPR records */
    3 Flags        fixed bin(31), /* General flags */
    3 PutMsgRecFields fixed bin(31), /* Flags indicating which MQPMPR
                                fields are present */
    3 RecsPresent  fixed bin(31), /* Number of MQOR records present */
```

MQDH – Language declarations

```
3 ObjectRecOffset fixed bin(31), /* Offset of first MQOR record from
                                start of MQDH */
3 PutMsgRecOffset fixed bin(31); /* Offset of first MQPMR record from
                                start of MQDH */
```

Visual Basic declaration

```
Type MQDH
  StrucId      As String*4 'Structure identifier'
  Version     As Long      'Structure version number'
  StrucLength  As Long      'Length of MQDH structure plus following'
                                'MQOR and MQPMR records'
  Encoding     As Long      'Numeric encoding of data that follows'
                                'the MQOR and MQPMR records'
  CodedCharSetId As Long    'Character set identifier of data that'
                                'follows the MQOR and MQPMR records'
  Format       As String*8 'Format name of data that follows the'
                                'MQOR and MQPMR records'
  Flags       As Long      'General flags'
  PutMsgRecFields As Long   'Flags indicating which MQPMR fields are'
                                'present'
  RecsPresent As Long      'Number of MQOR records present'
  ObjectRecOffset As Long   'Offset of first MQOR record from start'
                                'of MQDH'
  PutMsgRecOffset As Long   'Offset of first MQPMR record from start'
                                'of MQDH'
End Type
```

Chapter 7. MQDLH – Dead-letter header

The following table summarizes the fields in the structure.

Table 43. Fields in MQDLH

Field	Description	Page
<i>StrucId</i>	Structure identifier	89
<i>Version</i>	Structure version number	89
<i>Reason</i>	Reason message arrived on dead-letter queue	88
<i>DestQName</i>	Name of original destination queue	86
<i>DestQMgrName</i>	Name of original destination queue manager	85
<i>Encoding</i>	Numeric encoding of data that follows MQDLH	86
<i>CodedCharSetId</i>	Character set identifier of data that follows MQDLH	85
<i>Format</i>	Format name of data that follows MQDLH	86
<i>PutApplType</i>	Type of application that put message on dead-letter queue	87
<i>PutApplName</i>	Name of application that put message on dead-letter queue	86
<i>PutDate</i>	Date when message was put on dead-letter queue	87
<i>PutTime</i>	Time when message was put on dead-letter queue	87

Overview

Availability: Not Windows 3.1, Windows 95, Windows 98.

Purpose: The MQDLH structure describes the information that prefixes the application message data of messages on the dead-letter (undelivered-message) queue. A message can arrive on the dead-letter queue either because the queue manager or message channel agent has redirected it to the queue, or because an application has put the message directly on the queue.

Format name: MQFMT_DEAD_LETTER_HEADER.

Character set and encoding: The fields in the MQDLH structure are in the character set and encoding given by the *CodedCharSetId* and *Encoding* fields in the header structure that precedes MQDLH, or by those fields in the MQMD structure if the MQDLH is at the start of the application message data.

The character set must be one that has single-byte characters for the characters that are valid in queue names.

Usage: Applications that put messages directly on the dead-letter queue should prefix the message data with an MQDLH structure, and initialize the fields with appropriate values. However, the queue manager does not require that an MQDLH structure be present, or that valid values have been specified for the fields.

MQDLH – Dead-letter header

If a message is too long to put on the dead-letter queue, the application should consider doing one of the following:

- Truncate the message data to fit on the dead-letter queue.
- Record the message on auxiliary storage and place an exception report message on the dead-letter queue indicating this.
- Discard the message and return an error to its originator. If the message is (or might be) a critical message, this should be done only if it is known that the originator still has a copy of the message—for example, a message received by a message channel agent from a communication channel.

Which of the above is appropriate (if any) depends on the design of the application.

The queue manager performs special processing when a message which is a segment is put with an MQDLH structure at the front; see the description of the MQMDE structure for further details.

Putting messages on the dead-letter queue: When a message is put on the dead-letter queue, the MQMD structure used for the MQPUT or MQPUT1 call should be identical to the MQMD associated with the message (usually the MQMD returned by the MQGET call), with the exception of the following:

- The *CodedCharSetId* and *Encoding* fields must be set to whatever character set and encoding are used for fields in the MQDLH structure.
- The *Format* field must be set to MQFMT_DEAD_LETTER_HEADER to indicate that the data begins with a MQDLH structure.
- The context fields (*AccountingToken*, *ApplIdentityData*, *ApplOriginData*, *PutApplName*, *PutApplType*, *PutDate*, *PutTime*, *UserIdentifier*) should be set by using a context option appropriate to the circumstances:
 - An application putting on the dead-letter queue a message that is not related to any preceding message should use the MQPMO_DEFAULT_CONTEXT option; this causes the queue manager to set all of the context fields in the message descriptor to their default values.
 - A server application putting on the dead-letter queue a message it has just received should use the MQPMO_PASS_ALL_CONTEXT option, in order to preserve the original context information.
 - A server application putting on the dead-letter queue a *reply* to a message it has just received should use the MQPMO_PASS_IDENTITY_CONTEXT option; this preserves the identity information but sets the origin information to be that of the server application.
 - A message channel agent putting on the dead-letter queue a message it received from its communication channel should use the MQPMO_SET_ALL_CONTEXT option, to preserve the original context information.

In the MQDLH structure itself, the fields should be set as follows:

- The *CodedCharSetId*, *Encoding* and *Format* fields should be set to the values that describe the data that follows the MQDLH structure—usually the values from the original message descriptor.
- The context fields *PutApplType*, *PutApplName*, *PutDate*, and *PutTime* should be set to values appropriate to the application that is putting the message on the dead-letter queue; these values are not related to the original message.
- Other fields should be set as appropriate.

The application should ensure that all fields have valid values, and that character fields are padded with blanks to the defined length of the field; the character data should not be terminated prematurely by using a null character, because the queue manager does not convert the null and subsequent characters to blanks in the MQDLH structure.

Getting messages from the dead-letter queue: Applications that get messages from the dead-letter queue should verify that the messages begin with an MQDLH structure. The application can determine whether an MQDLH structure is present by examining the *Format* field in the message descriptor MQMD; if the field has the value MQFMT_DEAD_LETTER_HEADER, the message data begins with an MQDLH structure. Applications that get messages from the dead-letter queue should also be aware that such messages may have been truncated if they were originally too long for the queue.

Fields

The MQDLH structure contains the following fields; the fields are described in **alphabetic order**:

CodedCharSetId (MQLONG)

Character set identifier of data that follows MQDLH.

This specifies the character set identifier of the data that follows the MQDLH structure (usually the data from the original message); it does not apply to character data in the MQDLH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

MQCCSI_INHERIT

Inherit character-set identifier of this structure.

Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

MQCCSI_INHERIT cannot be used if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

This value is supported in the following environments: AIX, HP-UX, z/OS, OS/2, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

The initial value of this field is MQCCSI_UNDEFINED.

DestQMgrName (MQCHAR48)

Name of original destination queue manager.

This is the name of the queue manager that was the original destination for the message.

MQDLH – DestQMgrName field

The length of this field is given by `MQ_Q_MGR_NAME_LENGTH`. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

DestQName (MQCHAR48)

Name of original destination queue.

This is the name of the message queue that was the original destination for the message.

The length of this field is given by `MQ_Q_NAME_LENGTH`. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

Encoding (MQLONG)

Numeric encoding of data that follows MQDLH.

This specifies the numeric encoding of the data that follows the MQDLH structure (usually the data from the original message); it does not apply to numeric data in the MQDLH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is 0.

Format (MQCHAR8)

Format name of data that follows MQDLH.

This specifies the format name of the data that follows the MQDLH structure (usually the data from the original message).

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *Format* field in MQMD.

The length of this field is given by `MQ_FORMAT_LENGTH`. The initial value of this field is `MQFMT_NONE`.

PutAppName (MQCHAR28)

Name of application that put message on dead-letter (undelivered-message) queue.

The format of the name depends on the *PutApplType* field. See, also, the description of the *PutAppName* field in Chapter 10, “MQMD – Message descriptor”, on page 141.

If it is the queue manager that redirects the message to the dead-letter queue, *PutAppName* contains the first 28 characters of the queue-manager name, padded with blanks if necessary.

The length of this field is given by `MQ_PUT_APPL_NAME_LENGTH`. The initial value of this field is the null string in C, and 28 blank characters in other programming languages.

PutAppIType (MQLONG)

Type of application that put message on dead-letter (undelivered-message) queue.

This field has the same meaning as the *PutAppIType* field in the message descriptor MQMD (see Chapter 10, “MQMD – Message descriptor”, on page 141 for details).

If it is the queue manager that redirects the message to the dead-letter queue, *PutAppIType* has the value MQAT_QMGR.

The initial value of this field is 0.

PutDate (MQCHAR8)

Date when message was put on dead-letter (undelivered-message) queue.

The format used for the date when this field is generated by the queue manager is:

YYYYMMDD

where the characters represent:

YYYY year (four numeric digits)

MM month of year (01 through 12)

DD day of month (01 through 31)

Greenwich Mean Time (GMT) is used for the *PutDate* and *PutTime* fields, subject to the system clock being set accurately to GMT.

On OS/2, the queue manager uses the TZ environment variable to calculate GMT. For more information on setting this variable, refer to the *WebSphere MQ System Administration Guide*.

The length of this field is given by MQ_PUT_DATE_LENGTH. The initial value of this field is the null string in C, and 8 blank characters in other programming languages.

PutTime (MQCHAR8)

Time when message was put on the dead-letter (undelivered-message) queue.

The format used for the time when this field is generated by the queue manager is:

HHMMSSTH

where the characters represent (in order):

HH hours (00 through 23)

MM minutes (00 through 59)

SS seconds (00 through 59; see note below)

T tenths of a second (0 through 9)

H hundredths of a second (0 through 9)

Note: If the system clock is synchronized to a very accurate time standard, it is possible on rare occasions for 60 or 61 to be returned for the seconds in *PutTime*. This happens when leap seconds are inserted into the global time standard.

Greenwich Mean Time (GMT) is used for the *PutDate* and *PutTime* fields, subject to the system clock being set accurately to GMT.

MQDLH – PutTime field

On OS/2, the queue manager uses the TZ environment variable to calculate GMT. For more information on setting this variable, refer to the *WebSphere MQ System Administration Guide* book.

The length of this field is given by MQ_PUT_TIME_LENGTH. The initial value of this field is the null string in C, and 8 blank characters in other programming languages.

Reason (MQLONG)

Reason message arrived on dead-letter (undelivered-message) queue.

This identifies the reason why the message was placed on the dead-letter queue instead of on the original destination queue. It should be one of the MQFB_* or MQRC_* values (for example, MQRC_Q_FULL). See the description of the *Feedback* field in Chapter 10, “MQMD – Message descriptor”, on page 141 for details of the common MQFB_* values that can occur.

If the value is in the range MQFB_IMS_FIRST through MQFB_IMS_LAST, the actual IMS error code can be determined by subtracting MQFB_IMS_ERROR from the value of the *Reason* field.

Some MQFB_* values occur only in this field. They relate to repository messages, trigger messages, or transmission-queue messages that have been transferred to the dead-letter queue. These are:

MQFB_APPL_CANNOT_BE_STARTED

Application cannot be started.

An application processing a trigger message was unable to start the application named in the *AppId* field of the trigger message (see Chapter 21, “MQTM – Trigger message”, on page 291).

On z/OS, the CKTI CICS transaction is an example of an application that processes trigger messages.

MQFB_APPL_TYPE_ERROR

Application type error.

An application processing a trigger message was unable to start the application because the *AppType* field of the trigger message is not valid (see Chapter 21, “MQTM – Trigger message”, on page 291).

On z/OS, the CKTI CICS transaction is an example of an application that processes trigger messages.

MQFB_BIND_OPEN_CLUSRCVR_DEL

Cluster-receiver channel deleted.

The message was on the SYSTEM.CLUSTER.TRANSMIT.QUEUE intended for a cluster queue that had been opened with the MQOO_BIND_ON_OPEN option, but the remote cluster-receiver channel to be used to transmit the message to the destination queue was deleted before the message could be sent. Because MQOO_BIND_ON_OPEN was specified, only the channel selected when the queue was opened can be used to transmit the message. As this channel is not longer available, the message has been placed on the dead-letter queue.

MQFB_NOT_A_REPOSITORY_MSG

Message is not a repository message.

MQFB_STOPPED_BY_CHAD_EXIT

Message stopped by channel auto-definition exit.

MQFB_STOPPED_BY_MSG_EXIT

Message stopped by channel message exit.

MQFB_TM_ERROR

MQTM structure not valid or missing.

The *Format* field in MQMD specifies MQFMT_TRIGGER, but the message does not begin with a valid MQTM structure. For example, the *StrucId* mnemonic eye-catcher may not be valid, the *Version* may not be recognized, or the length of the trigger message may be insufficient to contain the MQTM structure.

On z/OS, the CKTI CICS transaction is an example of an application that processes trigger messages and can generate this feedback code.

MQFB_XMIT_Q_MSG_ERROR

Message on transmission queue not in correct format.

A message channel agent has found that a message on the transmission queue is not in the correct format. The message channel agent puts the message on the dead-letter queue using this feedback code.

The initial value of this field is MQRC_NONE.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQDLH_STRUC_ID

Identifier for dead-letter header structure.

For the C programming language, the constant MQDLH_STRUC_ID_ARRAY is also defined; this has the same value as MQDLH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQDLH_STRUC_ID.

Version (MQLONG)

Structure version number.

The value must be:

MQDLH_VERSION_1

Version number for dead-letter header structure.

The following constant specifies the version number of the current version:

MQDLH_CURRENT_VERSION

Current version of dead-letter header structure.

The initial value of this field is MQDLH_VERSION_1.

Initial values and language declarations

Table 44. Initial values of fields in MQDLH

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQDLH_STRUC_ID	'DLHb'
<i>Version</i>	MQDLH_VERSION_1	1
<i>Reason</i>	MQRC_NONE	0
<i>DestQName</i>	None	Null string or blanks
<i>DestQMgrName</i>	None	Null string or blanks
<i>Encoding</i>	None	0
<i>CodedCharSetId</i>	MQCCSI_UNDEFINED	0
<i>Format</i>	MQFMT_NONE	Blanks
<i>PutApplType</i>	None	0
<i>PutApplName</i>	None	Null string or blanks
<i>PutDate</i>	None	Null string or blanks
<i>PutTime</i>	None	Null string or blanks
Notes:		
<ol style="list-style-type: none"> 1. The symbol 'b' represents a single blank character. 2. The value 'Null string or blanks' denotes the null string in C, and blank characters in other programming languages. 3. In the C programming language, the macro variable MQDLH_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure: <pre>MQDLH MyDLH = {MQDLH_DEFAULT};</pre> 		

C declaration

```
typedef struct tagMQDLH MQDLH;
struct tagMQDLH {
    MQCHAR4  StrucId;          /* Structure identifier */
    MQLONG   Version;         /* Structure version number */
    MQLONG   Reason;         /* Reason message arrived on dead-letter
                             (undelivered-message) queue */
    MQCHAR48 DestQName;      /* Name of original destination queue */
    MQCHAR48 DestQMgrName;   /* Name of original destination queue
                             manager */
    MQLONG   Encoding;       /* Numeric encoding of data that follows
                             MQDLH */
    MQLONG   CodedCharSetId; /* Character set identifier of data that
                             follows MQDLH */
    MQCHAR8  Format;         /* Format name of data that follows
                             MQDLH */
    MQLONG   PutApplType;    /* Type of application that put message on
                             dead-letter (undelivered-message)
                             queue */
    MQCHAR28 PutApplName;    /* Name of application that put message on
                             dead-letter (undelivered-message)
                             queue */
    MQCHAR8  PutDate;       /* Date when message was put on dead-letter
                             (undelivered-message) queue */
};
```

MQDLH – Language declarations

```
MQCHAR8  PutTime;          /* Time when message was put on the
                             dead-letter (undelivered-message)
                             queue */
};
```

COBOL declaration

```
** MQDLH structure
10 MQDLH.
** Structure identifier
15 MQDLH-STRUCID          PIC X(4).
** Structure version number
15 MQDLH-VERSION         PIC S9(9) BINARY.
** Reason message arrived on dead-letter (undelivered-message) queue
15 MQDLH-REASON          PIC S9(9) BINARY.
** Name of original destination queue
15 MQDLH-DESTQNAME       PIC X(48).
** Name of original destination queue manager
15 MQDLH-DESTQMGRNAME    PIC X(48).
** Numeric encoding of data that follows MQDLH
15 MQDLH-ENCODING        PIC S9(9) BINARY.
** Character set identifier of data that follows MQDLH
15 MQDLH-CODEDCHARSETID PIC S9(9) BINARY.
** Format name of data that follows MQDLH
15 MQDLH-FORMAT          PIC X(8).
** Type of application that put message on dead-letter
** (undelivered-message) queue
15 MQDLH-PUTAPPLTYPE     PIC S9(9) BINARY.
** Name of application that put message on dead-letter
** (undelivered-message) queue
15 MQDLH-PUTAPPLNAME     PIC X(28).
** Date when message was put on dead-letter (undelivered-message)
** queue
15 MQDLH-PUTDATE         PIC X(8).
** Time when message was put on the dead-letter (undelivered-message)
** queue
15 MQDLH-PUTTIME         PIC X(8).
```

PL/I declaration

```
dc1
1 MQDLH based,
3 StrucId          char(4),          /* Structure identifier */
3 Version          fixed bin(31), /* Structure version number */
3 Reason           fixed bin(31), /* Reason message arrived on
                                   dead-letter (undelivered-message)
                                   queue */
3 DestQName        char(48),         /* Name of original destination
                                   queue */
3 DestQMgrName     char(48),         /* Name of original destination queue
                                   manager */
3 Encoding          fixed bin(31), /* Numeric encoding of data that
                                   follows MQDLH */
3 CodedCharSetId   fixed bin(31), /* Character set identifier of data
                                   that follows MQDLH */
3 Format            char(8),          /* Format name of data that follows
                                   MQDLH */
3 PutAppIType      fixed bin(31), /* Type of application that put
                                   message on dead-letter
                                   (undelivered-message) queue */
3 PutAppIName      char(28),         /* Name of application that put
                                   message on dead-letter
                                   (undelivered-message) queue */
3 PutDate          char(8),          /* Date when message was put on
                                   dead-letter (undelivered-message)
                                   queue */
```

MQDLH – Language declarations

```
3 PutTime      char(8);      /* Time when message was put on the
                             dead-letter (undelivered-message)
                             queue */
```

System/390 assembler declaration

```
MQDLH          DSECT
MQDLH_STRUCID  DS   CL4   Structure identifier
MQDLH_VERSION  DS   F     Structure version number
MQDLH_REASON   DS   F     Reason message arrived on dead-letter
*              (undelivered-message) queue
MQDLH_DESTQNAME DS   CL48 Name of original destination queue
MQDLH_DESTQMGRNAME DS   CL48 Name of original destination queue
*              manager
MQDLH_ENCODING DS   F     Numeric encoding of data that follows
*              MQDLH
MQDLH_CODEDCHARSETID DS   F Character set identifier of data that
*              follows MQDLH
MQDLH_FORMAT   DS   CL8   Format name of data that follows MQDLH
MQDLH_PUTAPPLTYPE DS   F   Type of application that put message on
*              dead-letter (undelivered-message) queue
MQDLH_PUTAPPLNAME DS   CL28 Name of application that put message on
*              dead-letter (undelivered-message) queue
MQDLH_PUTDATE  DS   CL8   Date when message was put on
*              dead-letter (undelivered-message) queue
MQDLH_PUTTIME  DS   CL8   Time when message was put on the
*              dead-letter (undelivered-message) queue
*
MQDLH_LENGTH   EQU   *-MQDLH
               ORG   MQDLH
MQDLH_AREA     DS   CL(MQDLH_LENGTH)
```

TAL declaration

```
STRUCT        MQDLH^DEF (*);
BEGIN
STRUCT        STRUCID;
BEGIN STRING BYTE [0:3]; END;
INT(32)       VERSION;
INT(32)       REASON;
STRUCT        DESTQNAME;
BEGIN STRING BYTE [0:47]; END;
STRUCT        DESTQMGRNAME;
BEGIN STRING BYTE [0:47]; END;
INT(32)       ENCODING;
INT(32)       CODEDCHARSETID;
STRUCT        FORMAT;
BEGIN STRING BYTE [0:7]; END;
INT(32)       PUTAPPLTYPE;
STRUCT        PUTAPPLNAME;
BEGIN STRING BYTE [0:27]; END;
STRUCT        PUTDATE;
BEGIN STRING BYTE [0:7]; END;
STRUCT        PUTTIME;
BEGIN STRING BYTE [0:7]; END;
END;
```

Visual Basic declaration

```
Type MQDLH
  StrucId      As String*4 'Structure identifier'
  Version      As Long     'Structure version number'
  Reason       As Long     'Reason message arrived on dead-letter'
                '(undelivered-message) queue'
  DestQName    As String*48 'Name of original destination queue'
  DestQMgrName As String*48 'Name of original destination queue'
                'manager'
```

MQDLH – Language declarations

```
Encoding      As Long      'Numeric encoding of data that follows'  
              'MQDLH'  
CodedCharSetId As Long      'Character set identifier of data that'  
              'follows MQDLH'  
Format        As String*8   'Format name of data that follows MQDLH'  
PutApplType   As Long      'Type of application that put message on'  
              'dead-letter (undelivered-message) queue'  
PutApplName   As String*28  'Name of application that put message on'  
              'dead-letter (undelivered-message) queue'  
PutDate       As String*8   'Date when message was put on dead-letter'  
              '(undelivered-message) queue'  
PutTime       As String*8   'Time when message was put on the'  
              'dead-letter (undelivered-message) queue'  
End Type
```

MQDLH – Language declarations

Chapter 8. MQGMO – Get-message options

The following table summarizes the fields in the structure.

Table 45. Fields in MQGMO

Field	Description	Page
<i>StrucId</i>	Structure identifier	126
<i>Version</i>	Structure version number	127
<i>Options</i>	Options that control the action of MQGET	100
<i>WaitInterval</i>	Wait interval	127
<i>Signal1</i>	Signal	125
<i>Signal2</i>	Signal identifier	126
<i>ResolvedQName</i>	Resolved name of destination queue	123
Note: The remaining fields are ignored if <i>Version</i> is less than MQGMO_VERSION_2.		
<i>MatchOptions</i>	Options controlling selection criteria used for MQGET	96
<i>GroupStatus</i>	Flag indicating whether message retrieved is in a group	96
<i>SegmentStatus</i>	Flag indicating whether message retrieved is a segment of a logical message	125
<i>Segmentation</i>	Flag indicating whether further segmentation is allowed for the message retrieved	124
<i>Reserved1</i>	Reserved	123
Note: The remaining fields are ignored if <i>Version</i> is less than MQGMO_VERSION_3.		
<i>MsgToken</i>	Message token	99
<i>ReturnedLength</i>	Length of message data returned (bytes)	124

Overview

Availability:

- Version 1: All
- Versions 2 and 3: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems

Purpose: The MQGMO structure allows the application to specify options that control how messages are removed from queues. The structure is an input/output parameter on the MQGET call.

Version: The current version of MQGMO is MQGMO_VERSION_3, but this version is not supported in all environments (see above). Applications that are intended to be portable between several environments must ensure that the required version of MQGMO is supported in all of the environments concerned. Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions that follow.

MQGMO – Get-message options

The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQGMO that is supported by the environment, but with the initial value of the *Version* field set to MQGMO_VERSION_1. To use fields that are not present in the version-1 structure, the application must set the *Version* field to the version number of the version required.

Character set and encoding: Data in MQGMO must be in the character set and encoding of the local queue manager; these are given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE, respectively. However, if the application is running as an MQ client, the structure must be in the character set and encoding of the client.

Fields

The MQGMO structure contains the following fields; the fields are described in **alphabetic order**:

GroupStatus (MQCHAR)

Flag indicating whether message retrieved is in a group.

It has one of the following values:

MQGS_NOT_IN_GROUP

Message is not in a group.

MQGS_MSG_IN_GROUP

Message is in a group, but is not the last in the group.

MQGS_LAST_MSG_IN_GROUP

Message is the last in the group.

This is also the value returned if the group consists of only one message.

This is an output field. The initial value of this field is MQGS_NOT_IN_GROUP. This field is ignored if *Version* is less than MQGMO_VERSION_2.

MatchOptions (MQLONG)

Options controlling selection criteria used for MQGET.

These options allow the application to choose which fields in the *MsgDesc* parameter will be used to select the message returned by the MQGET call. The application sets the required options in this field, and then sets the corresponding fields in the *MsgDesc* parameter to the values required for those fields. Only messages that have those values in the MQMD for the message are candidates for retrieval using that *MsgDesc* parameter on the MQGET call. Fields for which the corresponding match option is *not* specified are ignored when selecting the message to be returned. If no selection criteria are to be used on the MQGET call (that is, *any* message is acceptable), *MatchOptions* should be set to MQMO_NONE.

- On z/OS, the selection criteria that can be used may be restricted by the type of index used for the queue. See the *IndexType* queue attribute for further details.

If MQGMO_LOGICAL_ORDER is specified, only certain messages are eligible for return by the next MQGET call:

- If there is no current group or logical message, only messages that have *MsgSeqNumber* equal to 1 and *Offset* equal to 0 are eligible for return. In this

situation, one or more of the following match options can be used to select which of the eligible messages is the one actually returned:

MQMO_MATCH_MSG_ID
MQMO_MATCH_CORREL_ID
MQMO_MATCH_GROUP_ID

- If there *is* a current group or logical message, only the next message in the group or next segment in the logical message is eligible for return, and this cannot be altered by specifying MQMO_* options.

In both of the above cases, match options which are not applicable can still be specified, but the value of the relevant field in the *MsgDesc* parameter must match the value of the corresponding field in the message to be returned; the call fails with reason code MQRC_MATCH_OPTIONS_ERROR if this condition is not satisfied.

MatchOptions is ignored if either MQGMO_MSG_UNDER_CURSOR or MQGMO_BROWSE_MSG_UNDER_CURSOR is specified.

One or more of the following match options can be specified:

MQMO_MATCH_MSG_ID

Retrieve message with specified message identifier.

This option specifies that the message to be retrieved must have a message identifier that matches the value of the *MsgId* field in the *MsgDesc* parameter of the MQGET call. This match is in addition to any other matches that may apply (for example, the correlation identifier).

If this option is not specified, the *MsgId* field in the *MsgDesc* parameter is ignored, and any message identifier will match.

Note: The message identifier MQMI_NONE is a special value that matches *any* message identifier in the MQMD for the message. Therefore, specifying MQMO_MATCH_MSG_ID with MQMI_NONE is the same as *not* specifying MQMO_MATCH_MSG_ID.

MQMO_MATCH_CORREL_ID

Retrieve message with specified correlation identifier.

This option specifies that the message to be retrieved must have a correlation identifier that matches the value of the *CorrelId* field in the *MsgDesc* parameter of the MQGET call. This match is in addition to any other matches that may apply (for example, the message identifier).

If this option is not specified, the *CorrelId* field in the *MsgDesc* parameter is ignored, and any correlation identifier will match.

Note: The correlation identifier MQCI_NONE is a special value that matches *any* correlation identifier in the MQMD for the message. Therefore, specifying MQMO_MATCH_CORREL_ID with MQCI_NONE is the same as *not* specifying MQMO_MATCH_CORREL_ID.

MQMO_MATCH_GROUP_ID

Retrieve message with specified group identifier.

This option specifies that the message to be retrieved must have a group identifier that matches the value of the *GroupId* field in the *MsgDesc* parameter of the MQGET call. This match is in addition to any other matches that may apply (for example, the correlation identifier).

MQGMO – MatchOptions field

If this option is not specified, the *GroupId* field in the *MsgDesc* parameter is ignored, and any group identifier will match.

Note: The group identifier MQGI_NONE is a special value that matches *any* group identifier in the MQMD for the message. Therefore, specifying MQMO_MATCH_GROUP_ID with MQGI_NONE is the same as *not* specifying MQMO_MATCH_GROUP_ID.

MQMO_MATCH_MSG_SEQ_NUMBER

Retrieve message with specified message sequence number.

This option specifies that the message to be retrieved must have a message sequence number that matches the value of the *MsgSeqNumber* field in the *MsgDesc* parameter of the MQGET call. This match is in addition to any other matches that may apply (for example, the group identifier).

If this option is not specified, the *MsgSeqNumber* field in the *MsgDesc* parameter is ignored, and any message sequence number will match.

MQMO_MATCH_OFFSET

Retrieve message with specified offset.

This option specifies that the message to be retrieved must have an offset that matches the value of the *Offset* field in the *MsgDesc* parameter of the MQGET call. This match is in addition to any other matches that may apply (for example, the message sequence number).

If this option is not specified, the *Offset* field in the *MsgDesc* parameter is ignored, and any offset will match.

- This option is not supported on z/OS.

MQMO_MATCH_MSG_TOKEN

Retrieve message with specified message token.

This option specifies that the message to be retrieved must have a message token that matches the value of the *MsgToken* field in the MQGMO structure specified on the MQGET call.

This option can be specified only for queues that have an *IndexType* of MQIT_MSG_TOKEN. No other match options can be specified with MQMO_MATCH_MSG_TOKEN.

MQMO_MATCH_MSG_TOKEN cannot be specified with MQGMO_WAIT or MQGMO_SET_SIGNAL. If the application wants to wait for a message to arrive on a queue that has an *IndexType* of MQIT_MSG_TOKEN, MQMO_NONE must be specified.

If this option is not specified, the *MsgToken* field in MQGMO is ignored, and any message token will match.

- This option is supported only on z/OS.

If none of the options described above is specified, the following option can be used:

MQMO_NONE

No matches.

This option specifies that no matches are to be used in selecting the message to be returned; therefore, all messages on the queue are eligible for retrieval (but subject to control by the MQGMO_ALL_MSGS_AVAILABLE,

MQGMO – MatchOptions field

MQGMO_ALL_SEGMENTS_AVAILABLE, and MQGMO_COMPLETE_MSG options).

MQMO_NONE is defined to aid program documentation. It is not intended that this option be used with any other MQMO_* option, but as its value is zero, such use cannot be detected.

This is an input field. The initial value of this field is MQMO_MATCH_MSG_ID with MQMO_MATCH_CORREL_ID. This field is ignored if *Version* is less than MQGMO_VERSION_2.

Note: The initial value of the *MatchOptions* field is defined for compatibility with earlier MQSeries queue managers. However, when reading a series of messages from a queue without using selection criteria, this initial value requires the application to reset the *MsgId* and *CorrelId* fields to MQMI_NONE and MQCI_NONE prior to each MQGET call. The need to reset *MsgId* and *CorrelId* can be avoided by setting *Version* to MQGMO_VERSION_2, and *MatchOptions* to MQMO_NONE.

MsgToken (MQBYTE16)

Message token.

This is a byte string that is generated by the queue manager to identify a message uniquely. The message token is generated when the message is first placed on the queue manager, and remains with the message until the message is permanently removed from the queue manager. The message token persists across restarts of the queue manager. The token is local to the queue manager that generates it, and does not travel with the message when the message flows between queue managers. As a result, a particular message has a different message token on each of the queue managers that it visits.

Message tokens are supported in the following environments: z/OS.

For the MQGET call, *MsgToken* is one of the fields that can be used to select a particular message to be retrieved from the queue. Normally the MQGET call returns the next message on the queue, but if a message with a particular message token is required, this can be obtained by setting the *MsgToken* field to the value required and specifying the MQMO_MATCH_MSG_TOKEN option in the *MatchOptions* field.

Notes:

1. MQMO_MATCH_MSG_TOKEN can be specified only for queues that have an *IndexType* of MQIT_MSG_TOKEN.
2. No other MQMO_* options can be specified with MQMO_MATCH_MSG_TOKEN.

On return from an MQGET call, the *MsgToken* field is set to the message token of the message returned (if any). If the message does not have a message token, *MsgToken* is set to the following value:

MQMTOK_NONE

No message token.

The value is binary zero for the length of the field.

MQGMO – MsgToken field

For the C programming language, the constant MQMTOK_NONE_ARRAY is also defined; this has the same value as MQMTOK_NONE, but is an array of characters instead of a string.

This is an input/output field if MQMO_MATCH_MSG_TOKEN is specified, and an output field otherwise. The length of this field is given by MQ_MSG_TOKEN_LENGTH. The initial value of this field is MQMTOK_NONE. This field is ignored if *Version* is less than MQGMO_VERSION_3.

Options (MQLONG)

Options that control the action of MQGET.

Zero or more of the options described below can be specified. If more than one is required the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

Combinations of options that are not valid are noted; all other combinations are valid.

Wait options: The following options relate to waiting for messages to arrive on the queue:

MQGMO_WAIT

Wait for message to arrive.

The application is to wait until a suitable message arrives. The maximum time the application waits is specified in *WaitInterval*.

If MQGET requests are inhibited, or MQGET requests become inhibited while waiting, the wait is canceled and the call completes with MQCC_FAILED and reason code MQRC_GET_INHIBITED, regardless of whether there are suitable messages on the queue.

This option can be used with the MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT options.

If several applications are waiting on the same shared queue, the application, or applications, that are activated when a suitable message arrives are described below.

Note: In the description below, a *browse* MQGET call is one which specifies one of the browse options, but *not* MQGMO_LOCK; an MQGET call specifying the MQGMO_LOCK option is treated as a *nonbrowse* call.

- If one or more nonbrowse MQGET calls is waiting, but no browse MQGET calls are waiting, one is activated.
- If one or more browse MQGET calls is waiting, but no nonbrowse MQGET calls are waiting, all are activated.
- If one or more nonbrowse MQGET calls, and one or more browse MQGET calls are waiting, one nonbrowse MQGET call is activated, and none, some, or all of the browse MQGET calls. (The number of browse MQGET calls activated cannot be predicted, because it depends on the scheduling considerations of the operating system, and other factors.)

If more than one nonbrowse MQGET call is waiting on the same queue, only one is activated; in this situation the queue manager attempts to give priority to waiting nonbrowse calls in the following order:

MQGMO – Options field

1. Specific get-wait requests that can be satisfied only by certain messages, for example, ones with a specific *MsgId* or *CorrelId* (or both).
2. General get-wait requests that can be satisfied by any message.

The following points should be noted:

- Within the first category, no additional priority is given to more specific get-wait requests, for example those that specify both *MsgId* and *CorrelId*.
- Within either category, it cannot be predicted which application is selected. In particular, the application waiting longest is not necessarily the one selected.
- Path length, and priority-scheduling considerations of the operating system, can mean that a waiting application of lower operating system priority than expected retrieves the message.
- It may also happen that an application that is not waiting retrieves the message in preference to one that is.

On z/OS and Compaq NonStop Kernel, the following points apply:

- If it is desirable for the application to proceed with other work while waiting for the message to arrive, consider using the signal option (MQGMO_SET_SIGNAL) instead. However the signal option is environment specific, and so should not be used by applications which are intended to be portable between different environments.
- If there is more than one MQGET call waiting for the same message, with a mixture of wait and signal options, each waiting call is considered equally. It is an error to specify MQGMO_SET_SIGNAL with MQGMO_WAIT. It is also an error to specify this option with a queue handle for which a signal is outstanding.

On z/OS, the following point applies:

- If MQGMO_WAIT or MQGMO_SET_SIGNAL is specified for a queue that has an *IndexType* of MQIT_MSG_TOKEN, no selection criteria are permitted. This means that:
 - If a version-1 MQGMO is being used, the *MsgId* and *CorrelId* fields in the MQMD specified on the MQGET call must be set to MQMI_NONE and MQCI_NONE respectively.
 - If a version-2 or later MQGMO is being used, the *MatchOptions* field must be set to MQMO_NONE.

On Compaq NonStop Kernel, the following point applies:

- If an application specifies MQGET with MQGMO_SET_SIGNAL and a WaitInterval of 0, the MQGMO_SET_SIGNAL option is ignored and treated as an MQGET with MQGMO_NO_WAIT.

This means that an application must be prepared to receive MQRC_NO_MSG_AVAILABLE on an MQGET with MQGMO_SET_SIGNAL if the WaitInterval can ever be zero.

Applications receive a signal IPC only if:

- The application experiences MQRC_SIGNAL_REQUEST_ACCEPTED from the MQGET (indicating that a signal has been *posted*)
- The application has been able to process the file_open_system message and accept the signal IPC within the queue server's timeout

MQGMO – Options field

for signal delivery. This is 60s by default, but can be overridden for a queue server by specifying the MQQSSIGTIMEOUT PARAM in the environment of the queue server.

The queue manager logs the failure to deliver an IPC message to an application if it has not been able to open the process and send the IPC before the timeout expires. At this point the queue manager does not attempt delivery again. Applications should be resilient to this by not waiting indefinitely for an IPC signal.

MQGMO_WAIT is ignored if specified with MQGMO_BROWSE_MSG_UNDER_CURSOR or MQGMO_MSG_UNDER_CURSOR; no error is raised.

MQGMO_NO_WAIT

Return immediately if no suitable message.

The application is not to wait if no suitable message is available. This is the opposite of the MQGMO_WAIT option, and is defined to aid program documentation. It is the default if neither is specified.

MQGMO_SET_SIGNAL

Request signal to be set.

This option is used in conjunction with the *Signal1* and *Signal2* fields to allow applications to proceed with other work while waiting for a message to arrive, and also (if suitable operating system facilities are available) to wait for messages arriving on more than one queue.

Note: The MQGMO_SET_SIGNAL option is environment specific, and should not be used by applications that are intended to be portable.

If a currently available message satisfies the criteria specified in the message descriptor, or if a parameter error or other synchronous error is detected, the call completes in the same way as if this option had not been specified.

If no message satisfying the criteria specified in the message descriptor is currently available, control returns to the application without waiting for a message to arrive. The output fields in the message descriptor and the output parameters of the MQGET call are not set, other than the *CompCode* and *Reason* parameters (which are set to MQCC_WARNING and MQRC_SIGNAL_REQUEST_ACCEPTED respectively). When a suitable message arrives subsequently, the signal is delivered in a manner dependent on the environment:

- On z/OS, the signal is delivered by posting the ECB.
- On Compaq NonStop Kernel, an inter-process communication (IPC) message is sent to the \$RECEIVE queue of the process issuing the MQGET call.

The caller should then reissue the MQGET call to retrieve the message. The application can wait for this signal, using functions provided by the operating system.

If the operating system provides a multiple wait mechanism, the application can use this technique to wait for a message arriving on any one of several queues.

MQGMO – Options field

If a nonzero *WaitInterval* is specified, after this time the signal is delivered. The wait may also be canceled by the queue manager, in which case again the signal is delivered.

If more than one MQGET call has set a signal for the same message, the order in which applications are activated is the same as that described for MQGMO_WAIT.

If there is more than one MQGET call waiting for the same message, with a mixture of wait and signal options, each waiting call is considered equally.

Under certain conditions it is possible for a message to be retrieved by the MQGET call, *and* for a signal resulting from the arrival of the same message to be delivered. When a signal is delivered, an application must be prepared for no message to be available.

A queue handle can have no more than one signal request outstanding.

This option is not valid with any of the following options:

MQGMO_UNLOCK
MQGMO_WAIT

This option is supported in the following environments: z/OS, Compaq NonStop Kernel, Windows 95, Windows 98.

MQGMO_FAIL_IF QUIESCING

Fail if queue manager is quiescing.

This option forces the MQGET call to fail if the queue manager is in the quiescing state.

On z/OS, this option also forces the MQGET call to fail if the connection (for a CICS or IMS application) is in the quiescing state.

If this option is specified together with MQGMO_WAIT or MQGMO_SET_SIGNAL, and the wait or signal is outstanding at the time the queue manager enters the quiescing state:

- The wait is canceled and the call returns completion code MQCC_FAILED with reason code MQRC_Q_MGR QUIESCING or MQRC_CONNECTION QUIESCING.
- The signal is canceled with an environment-specific signal completion code.

On z/OS, the signal completes with event completion code MQEC_Q_MGR QUIESCING or MQEC_CONNECTION QUIESCING.

If MQGMO_FAIL_IF QUIESCING is not specified and the queue manager or connection enters the quiescing state, the wait or signal is not canceled.

On VSE/ESA, this option is not supported.

Syncpoint options: The following options relate to the participation of the MQGET call within a unit of work:

MQGMO_SYNCPOINT

Get message with syncpoint control.

The request is to operate within the normal unit-of-work protocols. The message is marked as being unavailable to other applications, but it is

MQGMO – Options field

deleted from the queue only when the unit of work is committed. The message is made available again if the unit of work is backed out.

If neither this option nor MQGMO_NO_SYNCPOINT is specified, the inclusion of the get request in unit-of-work protocols is determined by the environment:

- On z/OS, Compaq NonStop Kernel, and VSE/ESA, the get request is within a unit of work.
- In all other environments, the get request is not within a unit of work.

Because of these differences, an application which is intended to be portable should not allow this option to default; either MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT should be specified explicitly.

This option is not valid with any of the following options:

MQGMO_BROWSE_FIRST
MQGMO_BROWSE_MSG_UNDER_CURSOR
MQGMO_BROWSE_NEXT
MQGMO_LOCK
MQGMO_NO_SYNCPOINT
MQGMO_SYNCPOINT_IF_PERSISTENT
MQGMO_UNLOCK

MQGMO_SYNCPOINT_IF_PERSISTENT

Get message with syncpoint control if message is persistent.

The request is to operate within the normal unit-of-work protocols, but *only* if the message retrieved is persistent. A persistent message has the value MQPER_PERSISTENT in the *Persistence* field in MQMD.

- If the message is persistent, the queue manager processes the call as though the application had specified MQGMO_SYNCPOINT (see above for details).
- If the message is not persistent, the queue manager processes the call as though the application had specified MQGMO_NO_SYNCPOINT (see below for details).

This option is not valid with any of the following options:

MQGMO_BROWSE_FIRST
MQGMO_BROWSE_MSG_UNDER_CURSOR
MQGMO_BROWSE_NEXT
MQGMO_COMPLETE_MSG
MQGMO_MARK_SKIP_BACKOUT
MQGMO_NO_SYNCPOINT
MQGMO_SYNCPOINT
MQGMO_UNLOCK

This option is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows 98, Windows, plus WebSphere MQ clients connected to these systems.

MQGMO_NO_SYNCPOINT

Get message without syncpoint control.

The request is to operate outside the normal unit-of-work protocols. The message is deleted from the queue immediately (unless this is a browse request). The message cannot be made available again by backing out the unit of work.

This option is assumed if MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT is specified.

If neither this option nor MQGMO_SYNCPOINT is specified, the inclusion of the get request in unit-of-work protocols is determined by the environment:

- On z/OS, Compaq NonStop Kernel, and VSE/ESA, the get request is within a unit of work.
- In all other environments, the get request is not within a unit of work.

Because of these differences, an application which is intended to be portable should not allow this option to default; either MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT should be specified explicitly.

This option is not valid with any of the following options:

MQGMO_MARK_SKIP_BACKOUT
MQGMO_SYNCPOINT
MQGMO_SYNCPOINT_IF_PERSISTENT

On VSE/ESA, this option is not supported.

MQGMO_MARK_SKIP_BACKOUT

Mark the get request as skipping backout.

This option allows a unit of work to be backed out, but without reinstating on the queue the message that was marked with this option.

When an application requests the backout of a unit of work containing a get request, a message that was retrieved using this option is not restored to its previous state. (Other resource updates, however, are still backed out.) Instead, the message is treated as if it had been retrieved by a get request *without* this option, in a new unit of work started by the backout request.

This is useful if a message is retrieved by the application, but only after some resources have been changed does it become apparent that the unit of work cannot complete successfully. If this option is not specified, backing out the unit of work causes the message to be reinstated on the queue, so that the same sequence of events will occur when the message is next retrieved. However, if this option is specified on the original MQGET call, backing out the unit of work causes the updates to the other resources to be backed out, but the message is treated as if it had been retrieved under a new unit of work. The application can now perform appropriate error handling (such as sending a report message to the sender of the original message, or placing the original message on the dead-letter queue), and then commit the new unit of work. Committing the new unit of work causes the message to be removed permanently from the original queue.

MQGMO_MARK_SKIP_BACKOUT marks a single physical message. If the message belongs to a message group, the other messages in the group are not marked. Similarly, if the marked message is a segment of a logical message, the other segments in the logical message are not marked. Any

MQGMO – Options field

message in a group can be marked, but if messages are retrieved using MQGMO_LOGICAL_ORDER, it is advantageous to mark the *first* message in the group. When the unit of work is backed out, the first (marked) message is moved to the new unit of work, while the second and later messages in the group are reinstated on the queue. However, that group is no longer eligible for retrieval by an application using MQGMO_LOGICAL_ORDER, because the first message in the group is no longer on the queue. This prevents a second instance of the application inadvertently processing messages in that group. However, the first instance of the application can retrieve the second and later messages into the new unit of work using the MQGMO_LOGICAL_ORDER option as normal.

Occasionally it may be necessary to back out the new unit of work (for example, because the dead-letter queue is full and the message must not be discarded). Backing out the new unit of work reinstates the message on the original queue, which prevents the message being lost. However, in this situation processing cannot continue. After backing out the new unit of work, the application should inform the operator or administrator that there is an unrecoverable error, and then terminate.

This option has an effect only if the unit of work containing the get request is terminated by the application backing it out. (Such requests use calls or commands that depend on the environment.) This option has no effect if the unit of work containing the get request is backed out for any other reason (for example, because the transaction or system abends). In this situation, any message retrieved using this option is reinstated on the queue in the same way as messages retrieved without this option.

Notes:

1. If you have not applied IMS APAR PN60855 (or PN57124 for IMS V4), an IMS MPP or BMP application that backs out a unit of work containing a message retrieved with the MQGMO_MARK_SKIP_BACKOUT option must issue an MQ call (any MQ call will do) before committing or backing out the new unit of work.
2. A CICS application that backs out a unit of work containing a message retrieved with the MQGMO_MARK_SKIP_BACKOUT option must issue an MQ call (any MQ call will do) before committing or backing out the new unit of work.

Within a unit of work, there can be only one get request marked as skipping backout, as well as none or several unmarked get requests.

If this option is specified, MQGMO_SYNCPOINT must also be specified. MQGMO_MARK_SKIP_BACKOUT is not valid with any of the following options:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT
- MQGMO_LOCK
- MQGMO_NO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_UNLOCK

This option is supported only on z/OS.

Browse options: The following options relate to browsing messages on the queue:

MQGMO_BROWSE_FIRST

Browse from start of queue.

When a queue is opened with the MQOO_BROWSE option, a browse cursor is established, positioned logically before the first message on the queue. Subsequent MQGET calls specifying the MQGMO_BROWSE_FIRST, MQGMO_BROWSE_NEXT or MQGMO_BROWSE_MSG_UNDER_CURSOR option can be used to retrieve messages from the queue nondestructively. The browse cursor marks the position, within the messages on the queue, from which the next MQGET call with MQGMO_BROWSE_NEXT will search for a suitable message.

An MQGET call with MQGMO_BROWSE_FIRST causes the previous position of the browse cursor to be ignored. The first message on the queue that satisfies the conditions specified in the message descriptor is retrieved. The message remains on the queue, and the browse cursor is positioned on this message.

After this call, the browse cursor is positioned on the message that has been returned. If the message is removed from the queue before the next MQGET call with MQGMO_BROWSE_NEXT is issued, the browse cursor remains at the position in the queue that the message occupied, even though that position is now empty.

The MQGMO_MSG_UNDER_CURSOR option can subsequently be used with a nonbrowse MQGET call if required, to remove the message from the queue.

Note that the browse cursor is not moved by a nonbrowse MQGET call using the same *Hobj* handle. Nor is it moved by a browse MQGET call that returns a completion code of MQCC_FAILED, or a reason code of MQRC_TRUNCATED_MSG_FAILED.

The MQGMO_LOCK option can be specified together with this option, to cause the message that is browsed to be locked.

MQGMO_BROWSE_FIRST can be specified with any valid combination of the MQGMO_* and MQMO_* options that control the processing of messages in groups and segments of logical messages.

If MQGMO_LOGICAL_ORDER is specified, the messages are browsed in logical order. If that option is omitted, the messages are browsed in physical order. When MQGMO_BROWSE_FIRST is specified, it is possible to switch between logical order and physical order, but subsequent MQGET calls using MQGMO_BROWSE_NEXT must browse the queue in the same order as the most-recent call that specified MQGMO_BROWSE_FIRST for the queue handle.

The group and segment information that the queue manager retains for MQGET calls that browse messages on the queue is separate from the group and segment information that the queue manager retains for MQGET calls that remove messages from the queue. When MQGMO_BROWSE_FIRST is specified, the queue manager ignores the group and segment information for browsing, and scans the queue as though there were no current group and no current logical message. If the MQGET call is successful (completion code MQCC_OK or MQCC_WARNING), the group and segment information for browsing is

MQGMO – Options field

set to that of the message returned; if the call fails, the group and segment information remains the same as it was prior to the call.

This option is not valid with any of the following options:

- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT
- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_MSG_UNDER_CURSOR
- MQGMO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_UNLOCK

It is also an error if the queue was not opened for browse.

MQGMO_BROWSE_NEXT

Browse from current position in queue.

The browse cursor is advanced to the next message on the queue that satisfies the selection criteria specified on the MQGET call. The message is returned to the application, but remains on the queue.

After a queue has been opened for browse, the first browse call using the handle has the same effect whether it specifies the MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT option.

If the message is removed from the queue before the next MQGET call with MQGMO_BROWSE_NEXT is issued, the browse cursor logically remains at the position in the queue that the message occupied, even though that position is now empty.

Messages are stored on the queue in one of two ways:

- FIFO within priority (MQMDS_PRIORITY), or
- FIFO *regardless* of priority (MQMDS_FIFO)

The *MsgDeliverySequence* queue attribute indicates which method applies (see Chapter 40, “Attributes for queues”, on page 457 for details).

If the queue has a *MsgDeliverySequence* of MQMDS_PRIORITY, and a message arrives on the queue that is of a higher priority than the one currently pointed to by the browse cursor, that message will not be found during the current sweep of the queue using MQGMO_BROWSE_NEXT. It can only be found after the browse cursor has been reset with MQGMO_BROWSE_FIRST (or by reopening the queue).

The MQGMO_MSG_UNDER_CURSOR option can subsequently be used with a nonbrowse MQGET call if required, to remove the message from the queue.

Note that the browse cursor is not moved by nonbrowse MQGET calls using the same *Hobj* handle.

The MQGMO_LOCK option can be specified together with this option, to cause the message that is browsed to be locked.

MQGMO_BROWSE_NEXT can be specified with any valid combination of the MQGMO_* and MQMO_* options that control the processing of messages in groups and segments of logical messages.

If MQGMO_LOGICAL_ORDER is specified, the messages are browsed in logical order. If that option is omitted, the messages are browsed in

physical order. When MQGMO_BROWSE_FIRST is specified, it is possible to switch between logical order and physical order, but subsequent MQGET calls using MQGMO_BROWSE_NEXT must browse the queue in the same order as the most-recent call that specified MQGMO_BROWSE_FIRST for the queue handle. The call fails with reason code MQRC_INCONSISTENT_BROWSE if this condition is not satisfied.

Note: Special care is needed if an MQGET call is used to browse *beyond the end* of a message group (or logical message not in a group) when MQGMO_LOGICAL_ORDER is not specified. For example, if the last message in the group happens to *precede* the first message in the group on the queue, using MQGMO_BROWSE_NEXT to browse beyond the end of the group, specifying MQMO_MATCH_MSG_SEQ_NUMBER with *MsgSeqNumber* set to 1 (to find the first message of the next group) would return again the first message in the group already browsed. This could happen immediately, or a number of MQGET calls later (if there are intervening groups).

The possibility of an infinite loop can be avoided by opening the queue *twice* for browse:

- Use the first handle to browse only the first message in each group.
- Use the second handle to browse only the messages within a specific group.
- Use the MQMO_* options to move the second browse cursor to the position of the first browse cursor, before browsing the messages in the group.
- Do not use MQGMO_BROWSE_NEXT to browse beyond the end of a group.

The group and segment information that the queue manager retains for MQGET calls that browse messages on the queue is separate from the group and segment information that it retains for MQGET calls that remove messages from the queue.

This option is not valid with any of the following options:

MQGMO_BROWSE_FIRST
MQGMO_BROWSE_MSG_UNDER_CURSOR
MQGMO_MARK_SKIP_BACKOUT
MQGMO_MSG_UNDER_CURSOR
MQGMO_SYNCPOINT
MQGMO_SYNCPOINT_IF_PERSISTENT
MQGMO_UNLOCK

It is also an error if the queue was not opened for browse.

MQGMO_BROWSE_MSG_UNDER_CURSOR

Browse message under browse cursor.

This option causes the message pointed to by the browse cursor to be retrieved nondestructively, regardless of the MQMO_* options specified in the *MatchOptions* field in MQGMO.

The message pointed to by the browse cursor is the one that was last retrieved using either the MQGMO_BROWSE_FIRST or the MQGMO_BROWSE_NEXT option. The call fails if neither of these calls has

MQGMO – Options field

been issued for this queue since it was opened, or if the message that was under the browse cursor has since been retrieved destructively.

The position of the browse cursor is not changed by this call.

The MQGMO_MSG_UNDER_CURSOR option can subsequently be used with a nonbrowse MQGET call if required, to remove the message from the queue.

Note that the browse cursor is not moved by a nonbrowse MQGET call using the same *Hobj* handle. Nor is it moved by a browse MQGET call that returns a completion code of MQCC_FAILED, or a reason code of MQRC_TRUNCATED_MSG_FAILED.

If MQGMO_BROWSE_MSG_UNDER_CURSOR is specified *with* MQGMO_LOCK:

- If there is already a message locked, it must be the one under the cursor, so that is returned *without* unlocking and relocking it; the message remains locked.
- If there is no locked message, the message under the browse cursor (if there is one) is locked and returned to the application; if there is no message under the browse cursor the call fails.

If MQGMO_BROWSE_MSG_UNDER_CURSOR is specified *without* MQGMO_LOCK:

- If there is already a message locked, it must be the one under the cursor. This message is returned to the application *and then unlocked*. Because the message is now unlocked, there is no guarantee that it can be browsed again, or retrieved destructively (it may be retrieved destructively by another application getting messages from the queue).
- If there is no locked message, the message under the browse cursor (if there is one) is returned to the application; if there is no message under the browse cursor the call fails.

If MQGMO_COMPLETE_MSG is specified with MQGMO_BROWSE_MSG_UNDER_CURSOR, the browse cursor must identify a message whose *Offset* field in MQMD is zero. If this condition is not satisfied, the call fails with reason code MQRC_INVALID_MSG_UNDER_CURSOR.

The group and segment information that the queue manager retains for MQGET calls that browse messages on the queue is separate from the group and segment information that it retains for MQGET calls that remove messages from the queue.

This option is not valid with any of the following options:

MQGMO_BROWSE_FIRST
MQGMO_BROWSE_NEXT
MQGMO_MARK_SKIP_BACKOUT
MQGMO_MSG_UNDER_CURSOR
MQGMO_SYNCPOINT
MQGMO_SYNCPOINT_IF_PERSISTENT
MQGMO_UNLOCK

It is also an error if the queue was not opened for browse.

On VSE/ESA, this option is not supported.

MQGMO_MSG_UNDER_CURSOR

Get message under browse cursor.

This option causes the message pointed to by the browse cursor to be retrieved, regardless of the MQMO_* options specified in the *MatchOptions* field in MQGMO. The message is removed from the queue.

The message pointed to by the browse cursor is the one that was last retrieved using either the MQGMO_BROWSE_FIRST or the MQGMO_BROWSE_NEXT option.

If MQGMO_COMPLETE_MSG is specified with MQGMO_MSG_UNDER_CURSOR, the browse cursor must identify a message whose *Offset* field in MQMD is zero. If this condition is not satisfied, the call fails with reason code MQRC_INVALID_MSG_UNDER_CURSOR.

This option is not valid with any of the following options:

MQGMO_BROWSE_FIRST
MQGMO_BROWSE_MSG_UNDER_CURSOR
MQGMO_BROWSE_NEXT
MQGMO_UNLOCK

It is also an error if the queue was not opened both for browse and for input. If the browse cursor is not currently pointing to a retrievable message, an error is returned by the MQGET call.

Lock options: The following options relate to locking messages on the queue:

MQGMO_LOCK

Lock message.

This option locks the message that is browsed, so that the message becomes invisible to any other handle open for the queue. The option can be specified only if one of the following options is also specified:

MQGMO_BROWSE_FIRST
MQGMO_BROWSE_NEXT
MQGMO_BROWSE_MSG_UNDER_CURSOR

Only one message can be locked per queue handle, but this can be a logical message or a physical message:

- If MQGMO_COMPLETE_MSG is specified, all of the message segments that comprise the logical message are locked to the queue handle (provided that they are all present on the queue and available for retrieval).
- If MQGMO_COMPLETE_MSG is *not* specified, only a single physical message is locked to the queue handle. If this message happens to be a segment of a logical message, the locked segment prevents other applications using MQGMO_COMPLETE_MSG to retrieve or browse the logical message.

The locked message is always the one under the browse cursor, and the message can be removed from the queue by a later MQGET call that specifies the MQGMO_MSG_UNDER_CURSOR option. Other MQGET calls using the queue handle can also remove the message (for example, a call that specifies the message identifier of the locked message).

MQGMO – Options field

If the call returns completion code MQCC_FAILED, or MQCC_WARNING with reason code MQRC_TRUNCATED_MSG_FAILED, no message is locked.

If the application decides not to remove the message from the queue, the lock is released by:

- Issuing another MQGET call for this handle, with either MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT specified (with or without MQGMO_LOCK); the message is unlocked if the call completes with MQCC_OK or MQCC_WARNING, but remains locked if the call completes with MQCC_FAILED. However, the following exceptions apply:
 - The message *is not* unlocked if MQCC_WARNING is returned with MQRC_TRUNCATED_MSG_FAILED.
 - The message *is* unlocked if MQCC_FAILED is returned with MQRC_NO_MSG_AVAILABLE.

If MQGMO_LOCK is also specified, the message returned is locked. If MQGMO_LOCK is not specified, there is no locked message after the call.

If MQGMO_WAIT is specified, and no message is immediately available, the unlock on the original message occurs before the start of the wait (providing the call is otherwise free from error).

- Issuing another MQGET call for this handle, with MQGMO_BROWSE_MSG_UNDER_CURSOR (without MQGMO_LOCK); the message is unlocked if the call completes with MQCC_OK or MQCC_WARNING, but remains locked if the call completes with MQCC_FAILED. However, the following exception applies:
 - The message *is not* unlocked if MQCC_WARNING is returned with MQRC_TRUNCATED_MSG_FAILED.
- Issuing another MQGET call for this handle with MQGMO_UNLOCK.
- Issuing an MQCLOSE call for this handle (either explicitly, or implicitly by the application ending).

No special open option is required to specify this option, other than MQOO_BROWSE, which is needed in order to specify the accompanying browse option.

This option is not valid with any of the following options:

MQGMO_MARK_SKIP_BACKOUT
MQGMO_SYNCPOINT
MQGMO_SYNCPOINT_IF_PERSISTENT
MQGMO_UNLOCK

This option is not supported in the following environments: Windows 3.1, Windows 95, Windows 98.

MQGMO_UNLOCK

Unlock message.

The message to be unlocked must have been previously locked by an MQGET call with the MQGMO_LOCK option. If there is no message locked for this handle, the call completes with MQCC_WARNING and MQRC_NO_MSG_LOCKED.

MQGMO – Options field

The *MsgDesc*, *BufferLength*, *Buffer*, and *DataLength* parameters are not checked or altered if MQGMO_UNLOCK is specified. No message is returned in *Buffer*.

No special open option is required to specify this option (although MQOO_BROWSE is needed to issue the lock request in the first place).

This option is not valid with any options *except* the following:

MQGMO_NO_WAIT
MQGMO_NO_SYNCPOINT

Both of these options are assumed whether specified or not.

Message-data options: The following options relate to the processing of the message data when the message is read from the queue:

MQGMO_ACCEPT_TRUNCATED_MSG

Allow truncation of message data.

If the message buffer is too small to hold the complete message, this option allows the MQGET call to fill the buffer with as much of the message as the buffer can hold, issue a warning completion code, and complete its processing. This means:

- When browsing messages, the browse cursor is advanced to the returned message.
- When removing messages, the returned message is removed from the queue.
- Reason code MQRC_TRUNCATED_MSG_ACCEPTED is returned if no other error occurs.

Without this option, the buffer is still filled with as much of the message as it can hold, a warning completion code is issued, but processing is not completed. This means:

- When browsing messages, the browse cursor is not advanced.
- When removing messages, the message is not removed from the queue.
- Reason code MQRC_TRUNCATED_MSG_FAILED is returned if no other error occurs.

MQGMO_CONVERT

Convert message data.

This option requests that the application data in the message should be converted, to conform to the *CodedCharSetId* and *Encoding* values specified in the *MsgDesc* parameter on the MQGET call, before the data is copied to the *Buffer* parameter.

The *Format* field specified when the message was put is assumed by the conversion process to identify the nature of the data in the message. Conversion of the message data is by the queue manager for built-in formats, and by a user-written exit for other formats. See Appendix F, "Data conversion", on page 581 for details of the data-conversion exit.

- If conversion is performed successfully, the *CodedCharSetId* and *Encoding* fields specified in the *MsgDesc* parameter are unchanged on return from the MQGET call.
- If conversion cannot be performed successfully (but the MQGET call otherwise completes without error), the message data is returned

MQGMO – Options field

unconverted, and the *CodedCharSetId* and *Encoding* fields in *MsgDesc* are set to the values for the unconverted message. The completion code is MQCC_WARNING in this case.

In either case, therefore, these fields describe the character-set identifier and encoding of the message data that is returned in the *Buffer* parameter.

See the *Format* field described in Chapter 10, “MQMD – Message descriptor”, on page 141 for a list of format names for which the queue manager performs the conversion.

This option is not supported in the following environments: z/OS using CICS Version 2, VSE/ESA, Windows 3.1, Windows 95, Windows 98.

Group and segment options: The following options relate to the processing of messages in groups and segments of logical messages. These definitions may be of help in understanding the options:

Physical message

This is the smallest unit of information that can be placed on or removed from a queue; it often corresponds to the information specified or retrieved on a single MQPUT, MQPUT1, or MQGET call. Every physical message has its own message descriptor (MQMD). Generally, physical messages are distinguished by differing values for the message identifier (*MsgId* field in MQMD), although this is not enforced by the queue manager.

Logical message

This is a single unit of application information. In the absence of system constraints, a logical message would be the same as a physical message. But where logical messages are extremely large, system constraints may make it advisable or necessary to split a logical message into two or more physical messages, called *segments*.

A logical message that has been segmented consists of two or more physical messages that have the same nonnull group identifier (*GroupId* field in MQMD), and the same message sequence number (*MsgSeqNumber* field in MQMD). The segments are distinguished by differing values for the segment offset (*Offset* field in MQMD), which gives the offset of the data in the physical message from the start of the data in the logical message. Because each segment is a physical message, the segments in a logical message usually have differing message identifiers.

A logical message that has not been segmented, but for which segmentation has been permitted by the sending application, also has a nonnull group identifier, although in this case there is only one physical message with that group identifier if the logical message does not belong to a message group. Logical messages for which segmentation has been inhibited by the sending application have a null group identifier (MQGI_NONE), unless the logical message belongs to a message group.

Message group

This is a set of one or more logical messages that have the same nonnull group identifier. The logical messages in the group are distinguished by differing values for the message sequence number, which is an integer in the range 1 through n, where n is the number of logical messages in the group. If one or more of the logical messages is segmented, there will be more than n physical messages in the group.

MQGMO_LOGICAL_ORDER

Messages in groups and segments of logical messages are returned in logical order.

This option controls the order in which messages are returned by *successive* MQGET calls for the queue handle. The option must be specified on each of those calls in order to have an effect.

If MQGMO_LOGICAL_ORDER is specified for successive MQGET calls for the queue handle, messages in groups are returned in the order given by their message sequence numbers, and segments of logical messages are returned in the order given by their segment offsets. This order may be different from the order in which those messages and segments occur on the queue.

Note: Specifying MQGMO_LOGICAL_ORDER has no adverse consequences on messages that do not belong to groups and that are not segments. In effect, such messages are treated as though each belonged to a message group consisting of only one message. Thus it is perfectly safe to specify MQGMO_LOGICAL_ORDER when retrieving messages from queues that may contain a mixture of messages in groups, message segments, and unsegmented messages not in groups.

To return the messages in the required order, the queue manager retains the group and segment information between successive MQGET calls. This information identifies the current message group and current logical message for the queue handle, the current position within the group and logical message, and whether the messages are being retrieved within a unit of work. Because the queue manager retains this information, the application does not need to set the group and segment information prior to each MQGET call. Specifically, it means that the application does not need to set the *GroupId*, *MsgSeqNumber*, and *Offset* fields in MQMD. However, the application does need to set the MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT option correctly on each call.

When the queue is opened, there is no current message group and no current logical message. A message group becomes the current message group when a message that has the MQMF_MSG_IN_GROUP flag is returned by the MQGET call. With MQGMO_LOGICAL_ORDER specified on successive calls, that group remains the current group until a message is returned that has:

- MQMF_LAST_MSG_IN_GROUP without MQMF_SEGMENT (that is, the last logical message in the group is not segmented), or
- MQMF_LAST_MSG_IN_GROUP with MQMF_LAST_SEGMENT (that is, the message returned is the last segment of the last logical message in the group).

When such a message is returned, the message group is terminated, and on successful completion of that MQGET call there is no longer a current group. In a similar way, a logical message becomes the current logical message when a message that has the MQMF_SEGMENT flag is returned by the MQGET call, and that logical message is terminated when the message that has the MQMF_LAST_SEGMENT flag is returned.

If no selection criteria are specified, successive MQGET calls return (in the correct order) the messages for the first message group on the queue, then

MQGMO – Options field

the messages for the second message group, and so on, until there are no more messages available. It is possible to select the particular message groups returned by specifying one or more of the following options in the *MatchOptions* field:

MQMO_MATCH_MSG_ID
 MQMO_MATCH_CORREL_ID
 MQMO_MATCH_GROUP_ID

However, these options are effective only when there is no current message group or logical message; see the *MatchOptions* field described in Chapter 8, “MQGMO – Get-message options”, on page 95 for further details.

Table 46 shows the values of the *MsgId*, *CorrelId*, *GroupId*, *MsgSeqNumber*, and *Offset* fields that the queue manager looks for when attempting to find a message to return on the MQGET call. This applies both to removing messages from the queue, and browsing messages on the queue. The columns in the table have the following meanings; “Either” means “Yes” or “No”:

LOG ORD

Indicates whether the MQGMO_LOGICAL_ORDER option is specified on the call.

Cur grp

Indicates whether a current message group exists prior to the call.

Cur log msg

Indicates whether a current logical message exists prior to the call.

Other columns

Show the values that the queue manager looks for. “Previous” denotes the value returned for the field in the previous message for the queue handle.

Table 46. MQGET options relating to messages in groups and segments of logical messages

Options you specify	Group and log-msg status prior to call		Values the queue manager looks for				
	Cur grp	Cur log msg	<i>MsgId</i>	<i>CorrelId</i>	<i>GroupId</i>	<i>MsgSeqNumber</i>	<i>Offset</i>
LOG ORD							
Yes	No	No	Controlled by <i>MatchOptions</i>	Controlled by <i>MatchOptions</i>	Controlled by <i>MatchOptions</i>	1	0
Yes	No	Yes	Any message identifier	Any correlation identifier	Previous group identifier	1	Previous offset + previous segment length
Yes	Yes	No	Any message identifier	Any correlation identifier	Previous group identifier	Previous sequence number + 1	0
Yes	Yes	Yes	Any message identifier	Any correlation identifier	Previous group identifier	Previous sequence number	Previous offset + previous segment length
No	Either	Either	Controlled by <i>MatchOptions</i>	Controlled by <i>MatchOptions</i>	Controlled by <i>MatchOptions</i>	Controlled by <i>MatchOptions</i>	Controlled by <i>MatchOptions</i>

When multiple message groups are present on the queue and eligible for return, the groups are returned in the order determined by the position on the queue of the first segment of the first logical message in each group

(that is, the physical messages that have message sequence numbers of 1, and offsets of 0, determine the order in which eligible groups are returned).

The MQGMO_LOGICAL_ORDER option affects units of work as follows:

- If the first logical message or segment in a group is retrieved within a unit of work, all of the other logical messages and segments in the group must be retrieved within a unit of work, if the same queue handle is used. However, they need not be retrieved within the same unit of work. This allows a message group consisting of many physical messages to be split across two or more consecutive units of work for the queue handle.
- If the first logical message or segment in a group is *not* retrieved within a unit of work, none of the other logical messages and segments in the group can be retrieved within a unit of work, if the same queue handle is used.

If these conditions are not satisfied, the MQGET call fails with reason code MQRC_INCONSISTENT_UOW.

When MQGMO_LOGICAL_ORDER is specified, the MQGMO supplied on the MQGET call must not be less than MQGMO_VERSION_2, and the MQMD must not be less than MQMD_VERSION_2. If this condition is not satisfied, the call fails with reason code MQRC_WRONG_GMO_VERSION or MQRC_WRONG_MD_VERSION, as appropriate.

If MQGMO_LOGICAL_ORDER is *not* specified for successive MQGET calls for the queue handle, messages are returned without regard for whether they belong to message groups, or whether they are segments of logical messages. This means that messages or segments from a particular group or logical message may be returned out of order, or they may be intermingled with messages or segments from other groups or logical messages, or with messages that are not in groups and are not segments. In this situation, the particular messages that are returned by successive MQGET calls is controlled by the MQMO_* options specified on those calls (see the *MatchOptions* field described in Chapter 8, “MQGMO – Get-message options”, on page 95 for details of these options).

This is the technique that can be used to restart a message group or logical message in the middle, after a system failure has occurred. When the system restarts, the application can set the *GroupId*, *MsgSeqNumber*, *Offset*, and *MatchOptions* fields to the appropriate values, and then issue the MQGET call with MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT set as desired, but *without* specifying MQGMO_LOGICAL_ORDER. If this call is successful, the queue manager retains the group and segment information, and subsequent MQGET calls using that queue handle can specify MQGMO_LOGICAL_ORDER as normal.

The group and segment information that the queue manager retains for the MQGET call is separate from the group and segment information that it retains for the MQPUT call. In addition, the queue manager retains separate information for:

- MQGET calls that remove messages from the queue.
- MQGET calls that browse messages on the queue.

MQGMO – Options field

For any given queue handle, the application is free to mix MQGET calls that specify MQGMO_LOGICAL_ORDER with MQGET calls that do not, but the following points should be noted:

- If MQGMO_LOGICAL_ORDER is *not* specified, each successful MQGET call causes the queue manager to set the saved group and segment information to the values corresponding to the message returned; this replaces the existing group and segment information retained by the queue manager for the queue handle. Only the information appropriate to the action of the call (browse or remove) is modified.
- If MQGMO_LOGICAL_ORDER is *not* specified, the call does not fail if there is a current message group or logical message; the call may however succeed with an MQCC_WARNING completion code. Table 47 shows the various cases that can arise. In these cases, if the completion code is not MQCC_OK, the reason code is one of the following (as appropriate):
 - MQRC_INCOMPLETE_GROUP
 - MQRC_INCOMPLETE_MSG
 - MQRC_INCONSISTENT_UOW

Note: The queue manager does not check the group and segment information when browsing a queue, or when closing a queue that was opened for browse but not input; in those cases the completion code is always MQCC_OK (assuming no other errors).

Table 47. Outcome when MQGET or MQCLOSE call is not consistent with group and segment information

Current call is	Previous call was MQGET with MQGMO_LOGICAL_ORDER	Previous call was MQGET without MQGMO_LOGICAL_ORDER
MQGET with MQGMO_LOGICAL_ORDER	MQCC_FAILED	MQCC_FAILED
MQGET without MQGMO_LOGICAL_ORDER	MQCC_WARNING	MQCC_OK
MQCLOSE with an unterminated group or logical message	MQCC_WARNING	MQCC_OK

Applications that simply want to retrieve messages and segments in logical order are recommended to specify MQGMO_LOGICAL_ORDER, as this is the simplest option to use. This option relieves the application of the need to manage the group and segment information, because the queue manager manages that information. However, specialized applications may need more control than provided by the MQGMO_LOGICAL_ORDER option, and this can be achieved by not specifying that option. If this is done, the application must ensure that the *MsgId*, *CorrelId*, *GroupId*, *MsgSeqNumber*, and *Offset* fields in MQMD, and the MQMO_* options in *MatchOptions* in MQGMO, are set correctly, prior to each MQGET call.

For example, an application that wants to *forward* physical messages that it receives, without regard for whether those messages are in groups or segments of logical messages, should *not* specify MQGMO_LOGICAL_ORDER. This is because in a complex network with multiple paths between sending and receiving queue managers, the physical messages may arrive out of order. By specifying neither MQGMO_LOGICAL_ORDER, nor the corresponding

MQPMO_LOGICAL_ORDER on the MQPUT call, the forwarding application can retrieve and forward each physical message as soon as it arrives, without having to wait for the next one in logical order to arrive.

MQGMO_LOGICAL_ORDER can be specified with any of the other MQGMO_* options, and with various of the MQMO_* options in appropriate circumstances (see above).

- On z/OS, this option is supported for private and shared queues, but the queue must have an index type of MQIT_GROUP_ID. For shared queues, the CFSTRUCT object that the queue map to must be at CFLEVEL(3).
- On AIX, HP-UX, OS/2, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems, this option is supported for all local queues.

MQGMO_COMPLETE_MSG

Only complete logical messages are retrievable.

This option specifies that only a complete logical message can be returned by the MQGET call. If the logical message is segmented, the queue manager reassembles the segments and returns the complete logical message to the application; the fact that the logical message was segmented is not apparent to the application retrieving it.

Note: This is the only option that causes the queue manager to reassemble message segments. If not specified, segments are returned individually to the application if they are present on the queue (and they satisfy the other selection criteria specified on the MQGET call). Applications that do not wish to receive individual segments should therefore always specify MQGMO_COMPLETE_MSG.

To use this option, the application must provide a buffer which is big enough to accommodate the complete message, or specify the MQGMO_ACCEPT_TRUNCATED_MSG option.

If the queue contains segmented messages with some of the segments missing (perhaps because they have been delayed in the network and have not yet arrived), specifying MQGMO_COMPLETE_MSG prevents the retrieval of segments belonging to incomplete logical messages. However, those message segments still contribute to the value of the *CurrentQDepth* queue attribute; this means that there may be no retrievable logical messages, even though *CurrentQDepth* is greater than zero.

For *persistent* messages, the queue manager can reassemble the segments only within a unit of work:

- If the MQGET call is operating within a user-defined unit of work, that unit of work is used. If the call fails partway through the reassembly process, the queue manager reinstates on the queue any segments that were removed during reassembly. However, the failure does not prevent the unit of work being committed successfully.
- If the call is operating outside a user-defined unit of work, and there is no user-defined unit of work in existence, the queue manager creates a unit of work just for the duration of the call. If the call is successful, the queue manager commits the unit of work automatically (the application does not need to do this). If the call fails, the queue manager backs out the unit of work.

MQGMO – Options field

- If the call is operating outside a user-defined unit of work, but a user-defined unit of work *does* exist, the queue manager is unable to perform reassembly. If the message does not require reassembly, the call can still succeed. But if the message *does* require reassembly, the call fails with reason code MQRC_UOW_NOT_AVAILABLE.

For *nonpersistent* messages, the queue manager does not require a unit of work to be available in order to perform reassembly.

Each physical message that is a segment has its own message descriptor. For the segments constituting a single logical message, most of the fields in the message descriptor will be the same for all segments in the logical message – usually it is only the *MsgId*, *Offset*, and *MsgFlags* fields that differ between segments in the logical message. However, if a segment is placed on a dead-letter queue at an intermediate queue manager, the DLQ handler retrieves the message specifying the MQGMO_CONVERT option, and this may result in the character set or encoding of the segment being changed. If the DLQ handler successfully sends the segment on its way, the segment may have a character set or encoding that differs from the other segments in the logical message when the segment finally arrives at the destination queue manager.

A logical message consisting of segments in which the *CodedCharSetId* and/or *Encoding* fields differ cannot be reassembled by the queue manager into a single logical message. Instead, the queue manager reassembles and returns the first few consecutive segments at the start of the logical message that have the same character-set identifiers and encodings, and the MQGET call completes with completion code MQCC_WARNING and reason code MQRC_INCONSISTENT_CCSDS or MQRC_INCONSISTENT_ENCODINGS, as appropriate. This happens regardless of whether MQGMO_CONVERT is specified. To retrieve the remaining segments, the application must reissue the MQGET call without the MQGMO_COMPLETE_MSG option, retrieving the segments one by one. MQGMO_LOGICAL_ORDER can be used to retrieve the remaining segments in order.

It is also possible for an application which puts segments to set other fields in the message descriptor to values that differ between segments. However, there is no advantage in doing this if the receiving application uses MQGMO_COMPLETE_MSG to retrieve the logical message. When the queue manager reassembles a logical message, it returns in the message descriptor the values from the message descriptor for the *first* segment; the only exception is the *MsgFlags* field, which the queue manager sets to indicate that the reassembled message is the only segment.

If MQGMO_COMPLETE_MSG is specified for a report message, the queue manager performs special processing. The queue manager checks the queue to see if all of the report messages of that report type relating to the different segments in the logical message are present on the queue. If they are, they can be retrieved as a single message by specifying MQGMO_COMPLETE_MSG. For this to be possible, either the report messages must be generated by a queue manager or MCA which supports segmentation, or the originating application must request at least 100 bytes of message data (that is, the appropriate MQRO*_WITH_DATA or MQRO*_WITH_FULL_DATA options must be specified). If less than the

full amount of application data is present for a segment, the missing bytes are replaced by nulls in the report message returned.

If MQGMO_COMPLETE_MSG is specified with MQGMO_MSG_UNDER_CURSOR or MQGMO_BROWSE_MSG_UNDER_CURSOR, the browse cursor must be positioned on a message whose *Offset* field in MQMD has a value of 0. If this condition is not satisfied, the call fails with reason code MQRC_INVALID_MSG_UNDER_CURSOR.

MQGMO_COMPLETE_MSG implies MQGMO_ALL_SEGMENTS_AVAILABLE, which need not therefore be specified.

MQGMO_COMPLETE_MSG can be specified with any of the other MQGMO_* options apart from MQGMO_SYNCPOINT_IF_PERSISTENT, and with any of the MQMO_* options apart from MQMO_MATCH_OFFSET.

- On z/OS, this option is supported for private and shared queues, but the queue must have an index type of MQIT_GROUP_ID. For shared queues, the CFSTRUCT object that the queue map to must be at CFLEVEL(3).
- On AIX, HP-UX, OS/2, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems, this option is supported for all local queues.

MQGMO_ALL_MSGS_AVAILABLE

All messages in group must be available.

This option specifies that messages in a group become available for retrieval only when *all* messages in the group are available. If the queue contains message groups with some of the messages missing (perhaps because they have been delayed in the network and have not yet arrived), specifying MQGMO_ALL_MSGS_AVAILABLE prevents retrieval of messages belonging to incomplete groups. However, those messages still contribute to the value of the *CurrentQDepth* queue attribute; this means that there may be no retrievable message groups, even though *CurrentQDepth* is greater than zero. If there are no other messages that are retrievable, reason code MQRC_NO_MSG_AVAILABLE is returned after the specified wait interval (if any) has expired.

The processing of MQGMO_ALL_MSGS_AVAILABLE depends on whether MQGMO_LOGICAL_ORDER is also specified:

- If both options are specified, MQGMO_ALL_MSGS_AVAILABLE has an effect *only* when there is no current group or logical message. If there *is* a current group or logical message, MQGMO_ALL_MSGS_AVAILABLE is ignored. This means that MQGMO_ALL_MSGS_AVAILABLE can remain on when processing messages in logical order.
- If MQGMO_ALL_MSGS_AVAILABLE is specified without MQGMO_LOGICAL_ORDER, MQGMO_ALL_MSGS_AVAILABLE *always* has an effect. This means that the option must be turned off after the first message in the group has been removed from the queue, in order to be able to remove the remaining messages in the group.

Successful completion of an MQGET call specifying MQGMO_ALL_MSGS_AVAILABLE means that at the time that the

MQGMO – Options field

MQGET call was issued, all of the messages in the group were on the queue. However, be aware that other applications are still able to remove messages from the group (the group is not locked to the application that retrieves the first message in the group).

If this option is not specified, messages belonging to groups can be retrieved even when the group is incomplete.

MQGMO_ALL_MSGS_AVAILABLE implies MQGMO_ALL_SEGMENTS_AVAILABLE, which need not therefore be specified.

MQGMO_ALL_MSGS_AVAILABLE can be specified with any of the other MQGMO_* options, and with any of the MQMO_* options.

- On z/OS, this option is supported for private and shared queues, but the queue must have an index type of MQIT_GROUP_ID. For shared queues, the CFSTRUCT object that the queue map to must be at CFLEVEL(3).
- On AIX, HP-UX, OS/2, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems, this option is supported for all local queues.

MQGMO_ALL_SEGMENTS_AVAILABLE

All segments in a logical message must be available.

This option specifies that segments in a logical message become available for retrieval only when *all* segments in the logical message are available. If the queue contains segmented messages with some of the segments missing (perhaps because they have been delayed in the network and have not yet arrived), specifying MQGMO_ALL_SEGMENTS_AVAILABLE prevents retrieval of segments belonging to incomplete logical messages. However those segments still contribute to the value of the *CurrentQDepth* queue attribute; this means that there may be no retrievable logical messages, even though *CurrentQDepth* is greater than zero. If there are no other messages that are retrievable, reason code MQRC_NO_MSG_AVAILABLE is returned after the specified wait interval (if any) has expired.

The processing of MQGMO_ALL_SEGMENTS_AVAILABLE depends on whether MQGMO_LOGICAL_ORDER is also specified:

- If both options are specified, MQGMO_ALL_SEGMENTS_AVAILABLE has an effect *only* when there is no current logical message. If there *is* a current logical message, MQGMO_ALL_SEGMENTS_AVAILABLE is ignored. This means that MQGMO_ALL_SEGMENTS_AVAILABLE can remain on when processing messages in logical order.
- If MQGMO_ALL_SEGMENTS_AVAILABLE is specified without MQGMO_LOGICAL_ORDER, MQGMO_ALL_SEGMENTS_AVAILABLE *always* has an effect. This means that the option must be turned off after the first segment in the logical message has been removed from the queue, in order to be able to remove the remaining segments in the logical message.

If this option is not specified, message segments can be retrieved even when the logical message is incomplete.

While both MQGMO_COMPLETE_MSG and MQGMO_ALL_SEGMENTS_AVAILABLE require all segments to be available before any of them can be retrieved, the former returns the complete message, whereas the latter allows the segments to be retrieved one by one.

If MQGMO_ALL_SEGMENTS_AVAILABLE is specified for a report message, the queue manager performs special processing. The queue manager checks the queue to see if there is at least one report message for each of the segments that comprise the complete logical message. If there is, the MQGMO_ALL_SEGMENTS_AVAILABLE condition is satisfied. However, the queue manager does not check the *type* of the report messages present, and so there may be a mixture of report types in the report messages relating to the segments of the logical message. As a result, the success of MQGMO_ALL_SEGMENTS_AVAILABLE does not imply that MQGMO_COMPLETE_MSG will succeed. If there *is* a mixture of report types present for the segments of a particular logical message, those report messages must be retrieved one by one.

MQGMO_ALL_SEGMENTS_AVAILABLE can be specified with any of the other MQGMO_* options, and with any of the MQMO_* options.

- On z/OS, this option is supported for private and shared queues, but the queue must have an index type of MQIT_GROUP_ID. For shared queues, the CFSTRUCT object that the queue map to must be at CFLEVEL(3).
- On AIX, HP-UX, OS/2, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems, this option is supported for all local queues.

Default option: If none of the options described above is required, the following option can be used:

MQGMO_NONE

No options specified.

This value can be used to indicate that no other options have been specified; all options assume their default values. MQGMO_NONE is defined to aid program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

The initial value of the *Options* field is MQGMO_NO_WAIT.

Reserved1 (MQCHAR)

Reserved.

This is a reserved field. The initial value of this field is a blank character. This field is ignored if *Version* is less than MQGMO_VERSION_2.

ResolvedQName (MQCHAR48)

Resolved name of destination queue.

This is an output field which is set by the queue manager to the local name of the queue from which the message was retrieved, as defined to the local queue manager. This will be different from the name used to open the queue if:

MQGMO – ResolvedQName field

- An alias queue was opened (in which case, the name of the local queue to which the alias resolved is returned), or
- A model queue was opened (in which case, the name of the dynamic local queue is returned).

The length of this field is given by `MQ_Q_NAME_LENGTH`. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ReturnedLength (MQLONG)

Length of message data returned (bytes).

This is an output field that is set by the queue manager to the length in bytes of the message data returned by the `MQGET` call in the *Buffer* parameter. If the queue manager does not support this capability, *ReturnedLength* is set to the value `MQRL_UNDEFINED`.

When messages are converted between encodings or character sets, the message data can sometimes change size. On return from the `MQGET` call:

- If *ReturnedLength* is *not* `MQRL_UNDEFINED`, the number of bytes of message data returned is given by *ReturnedLength*.
- If *ReturnedLength* has the value `MQRL_UNDEFINED`, the number of bytes of message data returned is usually given by the smaller of *BufferLength* and *DataLength*, but can be *less than* this if the `MQGET` call completes with reason code `MQRC_TRUNCATED_MSG_ACCEPTED`. If this happens, the insignificant bytes in the *Buffer* parameter are set to nulls.

The following special value is defined:

MQRL_UNDEFINED

Length of returned data not defined.

On z/OS, the value returned for the *ReturnedLength* field is always `MQRL_UNDEFINED`.

The initial value of this field is `MQRL_UNDEFINED`. This field is ignored if *Version* is less than `MQGMO_VERSION_3`.

Segmentation (MQCHAR)

Flag indicating whether further segmentation is allowed for the message retrieved.

It has one of the following values:

MQSEG_INHIBITED

Segmentation not allowed.

MQSEG_ALLOWED

Segmentation allowed.

On z/OS, the queue manager always sets this field to `MQSEG_INHIBITED`.

This is an output field. The initial value of this field is `MQSEG_INHIBITED`. This field is ignored if *Version* is less than `MQGMO_VERSION_2`.

SegmentStatus (MQCHAR)

Flag indicating whether message retrieved is a segment of a logical message.

It has one of the following values:

MQSS_NOT_A_SEGMENT

Message is not a segment.

MQSS_SEGMENT

Message is a segment, but is not the last segment of the logical message.

MQSS_LAST_SEGMENT

Message is the last segment of the logical message.

This is also the value returned if the logical message consists of only one segment.

On z/OS, the queue manager always sets this field to MQSS_NOT_A_SEGMENT.

This is an output field. The initial value of this field is MQSS_NOT_A_SEGMENT. This field is ignored if *Version* is less than MQGMO_VERSION_2.

Signal1 (MQLONG)

Signal.

This is an input field that is used only in conjunction with the MQGMO_SET_SIGNAL option; it identifies a signal that is to be delivered when a message is available.

Note: The data type and usage of this field are determined by the environment; for this reason, signals should not be used by applications which are intended to be portable between different environments.

- On z/OS, this field must contain the address of an Event Control Block (ECB). The ECB must be cleared by the application before the MQGET call is issued. The storage containing the ECB must not be freed until the queue is closed. The ECB is posted by the queue manager with one of the signal completion codes described below. These completion codes are set in bits 2 through 31 of the ECB—the area defined in the z/OS mapping macro IHAECB as being for a user completion code.
- On Compaq NonStop Kernel, this field must contain an application tag. The tag allows the application to associate the eventual inter-process communication (IPC) message sent to the application's \$RECEIVE queue with a particular MQGET call.
- In all other environments, this is a reserved field; its value is not significant.

The signal completion codes are:

MQEC_MSG_ARRIVED

Message has arrived.

A suitable message has arrived on the queue. This message has not been reserved for the caller; a second MQGET request must be issued, but note that another application might retrieve the message before the second request is made.

MQEC_WAIT_INTERVAL_EXPIRED

Requested wait period has expired.

MQGMO – Signal1 field

The specified *WaitInterval* has expired without a suitable message arriving.

MQEC_WAIT_CANCELED

Requested wait period has been canceled.

The wait was canceled for an indeterminate reason (such as the queue manager terminating, or the queue being disabled). The request must be reissued if further diagnosis is required.

MQEC_Q_MGR QUIESCING

Queue manager quiescing.

The wait was canceled because the queue manager has entered the quiescing state (MQGMO_FAIL_IF QUIESCING was specified on the MQGET call).

MQEC_CONNECTION QUIESCING

Connection quiescing.

The wait was canceled because the connection has entered the quiescing state (MQGMO_FAIL_IF QUIESCING was specified on the MQGET call).

The initial value of this field is determined by the environment:

- On z/OS, the initial value is the null pointer.
- In all other environments, the initial value is 0.

Signal2 (MQLONG)

Signal identifier.

This is an input field that is used only in conjunction with the MQGMO_SET_SIGNAL option. The data type and usage of this field are determined by the environment:

- On Windows 95, Windows 98, this field contains the identifier of a Windows message that is sent to the application window (as specified by the *Signal1* field) to signal that a suitable message has arrived. The Windows call RegisterWindowMessage should be used to obtain a suitable identifier.
- In all other environments, this is a reserved field; its value is not significant.

The initial value of this field is 0.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQGMO_STRUC_ID

Identifier for get-message options structure.

For the C programming language, the constant MQGMO_STRUC_ID_ARRAY is also defined; this has the same value as MQGMO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQGMO_STRUC_ID.

Version (MQLONG)

Structure version number.

The value must be one of the following:

MQGMO_VERSION_1

Version-1 get-message options structure.

This version is supported in all environments.

MQGMO_VERSION_2

Version-2 get-message options structure.

This version is supported in the following environments: AIX, HP-UX, z/OS, OS/2, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

MQGMO_VERSION_3

Version-3 get-message options structure.

This version is supported in the following environments: AIX, HP-UX, z/OS, OS/2, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQGMO_CURRENT_VERSION

Current version of get-message options structure.

This is always an input field. The initial value of this field is MQGMO_VERSION_1.

WaitInterval (MQLONG)

Wait interval.

This is the approximate time, expressed in milliseconds, that the MQGET call waits for a suitable message to arrive (that is, a message satisfying the selection criteria specified in the *MsgDesc* parameter of the MQGET call; see the *MsgId* field described in Chapter 10, “MQMD – Message descriptor”, on page 141 for more details). If no suitable message has arrived after this time has elapsed, the call completes with MQCC_FAILED and reason code MQRC_NO_MSG_AVAILABLE.

On z/OS, the period of time that the MQGET call actually waits is affected by system loading and work-scheduling considerations, and can vary between the value specified for *WaitInterval* and approximately 250 milliseconds greater than *WaitInterval*.

WaitInterval is used in conjunction with the MQGMO_WAIT or MQGMO_SET_SIGNAL option. It is ignored if neither of these is specified. If one of these is specified, *WaitInterval* must be greater than or equal to zero, or the following special value:

MQWI_UNLIMITED

Unlimited wait interval.

The initial value of this field is 0.

Initial values and language declarations

Table 48. Initial values of fields in MQGMO

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQGMO_STRUC_ID	'GMOb'
<i>Version</i>	MQGMO_VERSION_1	1
<i>Options</i>	MQGMO_NO_WAIT	0
<i>WaitInterval</i>	None	0
<i>Signal1</i>	None	Null pointer on z/OS; 0 otherwise
<i>Signal2</i>	None	0
<i>ResolvedQName</i>	None	Null string or blanks
<i>MatchOptions</i>	MQMO_MATCH_MSG_ID + MQMO_MATCH_CORREL_ID	3
<i>GroupStatus</i>	MQGS_NOT_IN_GROUP	'b'
<i>SegmentStatus</i>	MQSS_NOT_A_SEGMENT	'b'
<i>Segmentation</i>	MQSEG_INHIBITED	'b'
<i>Reserved1</i>	None	'b'
<i>MsgToken</i>	MQMTOK_NONE	Nulls
<i>ReturnedLength</i>	MQRL_UNDEFINED	-1

Notes:

1. The symbol 'b' represents a single blank character.
2. The value 'Null string or blanks' denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQGMO_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQGMO MyGMO = {MQGMO_DEFAULT};
```

C declaration

```
typedef struct tagMQGMO MQGMO;
struct tagMQGMO {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQLONG    Options;          /* Options that control the action of
                                MQGET */
    MQLONG    WaitInterval;     /* Wait interval */
    MQLONG    Signal1;          /* Signal */
    MQLONG    Signal2;          /* Signal identifier */
    MQCHAR48  ResolvedQName;    /* Resolved name of destination queue */
    MQLONG    MatchOptions;     /* Options controlling selection criteria
                                used for MQGET */
    MQCHAR    GroupStatus;      /* Flag indicating whether message
                                retrieved is in a group */
    MQCHAR    SegmentStatus;    /* Flag indicating whether message
                                retrieved is a segment of a logical
                                message */
    MQCHAR    Segmentation;     /* Flag indicating whether further
                                segmentation is allowed for the message
                                retrieved */
};
```

MQGMO – Language declarations

```
MQCHAR    Reserved1;      /* Reserved */
MQBYTE16  MsgToken;      /* Message token */
MQLONG    ReturnedLength; /* Length of message data returned
                          (bytes) */
};
```

- On z/OS, the *Signal1* field is declared as PMQLONG.

COBOL declaration

```
** MQGMO structure
10 MQGMO.
** Structure identifier
15 MQGMO-STRUCID      PIC X(4).
** Structure version number
15 MQGMO-VERSION     PIC S9(9) BINARY.
** Options that control the action of MQGET
15 MQGMO-OPTIONS     PIC S9(9) BINARY.
** Wait interval
15 MQGMO-WAITINTERVAL PIC S9(9) BINARY.
** Signal
15 MQGMO-SIGNAL1     PIC S9(9) BINARY.
** Signal identifier
15 MQGMO-SIGNAL2     PIC S9(9) BINARY.
** Resolved name of destination queue
15 MQGMO-RESOLVEDQNAME PIC X(48).
** Options controlling selection criteria used for MQGET
15 MQGMO-MATCHOPTIONS PIC S9(9) BINARY.
** Flag indicating whether message retrieved is in a group
15 MQGMO-GROUPSTATUS PIC X.
** Flag indicating whether message retrieved is a segment of a
** logical message
15 MQGMO-SEGMENTSTATUS PIC X.
** Flag indicating whether further segmentation is allowed for the
** message retrieved
15 MQGMO-SEGMENTATION PIC X.
** Reserved
15 MQGMO-RESERVED1   PIC X.
** Message token
15 MQGMO-MSGTOKEN    PIC X(16).
** Length of message data returned (bytes)
15 MQGMO-RETURNEDLENGTH PIC S9(9) BINARY.
```

- On z/OS, the *Signal1* field is declared as POINTER.

PL/I declaration

```
dc1
1 MQGMO based,
3 StrucId      char(4),      /* Structure identifier */
3 Version      fixed bin(31), /* Structure version number */
3 Options      fixed bin(31), /* Options that control the action of
                             MQGET */
3 WaitInterval fixed bin(31), /* Wait interval */
3 Signal1      fixed bin(31), /* Signal */
3 Signal2      fixed bin(31), /* Signal identifier */
3 ResolvedQName char(48),    /* Resolved name of destination
                             queue */
3 MatchOptions fixed bin(31), /* Options controlling selection
                             criteria used for MQGET */
3 GroupStatus  char(1),      /* Flag indicating whether message
                             retrieved is in a group */
3 SegmentStatus char(1),    /* Flag indicating whether message
                             retrieved is a segment of a logical
                             message */
3 Segmentation char(1),    /* Flag indicating whether further
                             segmentation is allowed for the
                             message retrieved */
```

MQGMO – Language declarations

```
3 Reserved1      char(1),      /* Reserved */
3 MsgToken       char(16),     /* Message token */
3 ReturnedLength fixed bin(31); /* Length of message data returned
                               (bytes) */
```

- On z/OS, the *Signal1* field is declared as pointer.

System/390 assembler declaration

```
MQGMO          DSECT
MQGMO_STRUCID  DS  CL4  Structure identifier
MQGMO_VERSION  DS  F    Structure version number
MQGMO_OPTIONS  DS  F    Options that control the action of
*              MQGET
MQGMO_WAITINTERVAL DS  F    Wait interval
MQGMO_SIGNAL1  DS  F    Signal
MQGMO_SIGNAL2  DS  F    Signal identifier
MQGMO_RESOLVEDQNAME DS  CL48 Resolved name of destination queue
MQGMO_MATCHOPTIONS DS  F    Options controlling selection criteria
*              used for MQGET
MQGMO_GROUPSTATUS DS  CL1  Flag indicating whether message
*              retrieved is in a group
MQGMO_SEGMENTSTATUS DS  CL1  Flag indicating whether message
*              retrieved is a segment of a logical
*              message
MQGMO_SEGMENTATION DS  CL1  Flag indicating whether further
*              segmentation is allowed for the message
*              retrieved
MQGMO_RESERVED1 DS  CL1  Reserved
MQGMO_MSGTOKEN  DS  XL16  Message token
MQGMO_RETURNEDLENGTH DS  F    Length of message data returned (bytes)
*
MQGMO_LENGTH   EQU  *-MQGMO
               ORG  MQGMO
MQGMO_AREA     DS  CL(MQGMO_LENGTH)
```

TAL declaration

```
STRUCT      MQGMO^DEF (*);
BEGIN
  STRUCT      STRUCID;
  BEGIN STRING BYTE [0:3]; END;
  INT(32)     VERSION;
  INT(32)     OPTIONS;
  INT(32)     WAITINTERVAL;
  INT(32)     SIGNAL1;
  INT(32)     SIGNAL2;
  STRUCT      RESOLVEDQNAME;
  BEGIN STRING BYTE [0:47]; END;
  END;
```

Visual Basic declaration

```
Type MQGMO
  StrucId      As String*4  'Structure identifier'
  Version      As Long      'Structure version number'
  Options      As Long      'Options that control the action of MQGET'
  WaitInterval As Long      'Wait interval'
  Signal1      As Long      'Signal'
  Signal2      As Long      'Signal identifier'
  ResolvedQName As String*48 'Resolved name of destination queue'
  MatchOptions As Long      'Options controlling selection criteria'
  'used for MQGET'
  GroupStatus  As String*1  'Flag indicating whether message'
  'retrieved is in a group'
  SegmentStatus As String*1 'Flag indicating whether message'
  'retrieved is a segment of a logical'
```

MQGMO – Language declarations

```
Segmentation  As String*1  'message'
                                     'Flag indicating whether further'
                                     'segmentation is allowed for the message'
                                     'retrieved'
Reserved1     As String*1  'Reserved'
MsgToken      As MQBYTE16  'Message token'
ReturnedLength As Long      'Length of message data returned (bytes)'
End Type
```

MQGMO – Language declarations

Chapter 9. MQIIH – IMS information header

The following table summarizes the fields in the structure.

Table 49. Fields in MQIIH

Field	Description	Page
<i>StrucId</i>	Structure identifier	136
<i>Version</i>	Structure version number	138
<i>StrucLength</i>	Length of MQIIH structure	137
<i>Encoding</i>	Reserved	135
<i>CodedCharSetId</i>	Reserved	134
<i>Format</i>	MQ format name of data that follows MQIIH	135
<i>Flags</i>	Flags	135
<i>LTermOverride</i>	Logical terminal override	135
<i>MFSMapName</i>	Message format services map name	136
<i>ReplyToFormat</i>	MQ format name of reply message	136
<i>Authenticator</i>	RACF™ password or passticket	134
<i>TranInstanceId</i>	Transaction instance identifier	137
<i>TranState</i>	Transaction state	137
<i>CommitMode</i>	Commit mode	134
<i>SecurityScope</i>	Security scope	136
<i>Reserved</i>	Reserved	136

Overview

Availability: Not Windows 3.1, Windows 95, Windows 98.

Note: IMS connection is not available when using the reduced function form of WebSphere MQ for z/OS supplied with WebSphere Application Server.

Purpose: The MQIIH structure describes the information that must be present at the start of a message sent to the IMS bridge through WebSphere MQ for z/OS.

Format name: MQFMT_IMS.

Character set and encoding: Special conditions apply to the character set and encoding used for the MQIIH structure and application message data:

- Applications that connect to the queue manager that owns the IMS bridge queue must provide an MQIIH structure that is in the character set and encoding of the queue manager. This is because data conversion of the MQIIH structure is not performed in this case.
- Applications that connect to other queue managers can provide an MQIIH structure that is in any of the supported character sets and encodings; conversion of the MQIIH is performed by the receiving message channel agent connected to the queue manager that owns the IMS bridge queue.

MQIIH – IMS information header

Note: There is one exception to this. If the queue manager that owns the IMS bridge queue is using CICS for distributed queuing, the MQIIH must be in the character set and encoding of the queue manager that owns the IMS bridge queue.

- The application message data following the MQIIH structure must be in the same character set and encoding as the MQIIH structure. The *CodedCharSetId* and *Encoding* fields in the MQIIH structure cannot be used to specify the character set and encoding of the application message data.

A data-conversion exit must be provided by the user to convert the application message data if the data is not one of the built-in formats supported by the queue manager.

Fields

The MQIIH structure contains the following fields; the fields are described in **alphabetic order**:

Authenticator (MQCHAR8)

RACF password or passticket.

This is optional; if specified, it is used with the user ID in the MQMD security context to build a Utoken that is sent to IMS to provide a security context. If it is not specified, the user ID is used without verification. This depends on the setting of the RACF switches, which may require an authenticator to be present.

This is ignored if the first byte is blank or null. The following special value may be used:

MQIAUT_NONE

No authentication.

For the C programming language, the constant `MQIAUT_NONE_ARRAY` is also defined; this has the same value as `MQIAUT_NONE`, but is an array of characters instead of a string.

The length of this field is given by `MQ_AUTHENTICATOR_LENGTH`. The initial value of this field is `MQIAUT_NONE`.

CodedCharSetId (MQLONG)

Reserved.

This is a reserved field; its value is not significant. The initial value of this field is 0.

CommitMode (MQCHAR)

Commit mode.

See the *OTMA Reference* for more information about IMS commit modes. The value must be one of the following:

MQICM_COMMIT_THEN_SEND

Commit then send.

This mode implies double queuing of output, but shorter region occupancy times. Fast-path and conversational transactions cannot run with this mode.

MQICM_SEND_THEN_COMMIT

Send then commit.

| Any IMS transaction initiated as a result of commitmode
 | MQICM_SEND_THEN_COMMIT will be forced to run in RESPONSE mode
 | regardless of how the transaction is defined in the IMS system definition
 | (MSGTYPE parameter in the TRANSACT macro). This also applies to transactions
 | initiated by means of a transaction switch.

The initial value of this field is MQICM_COMMIT_THEN_SEND.

Encoding (MQLONG)

Reserved.

This is a reserved field; its value is not significant. The initial value of this field is 0.

Flags (MQLONG)

Flags.

The value must be:

MQIIH_NONE

No flags.

MQIIH_PASS_EXPIRATION

The reply message contains:

- The same expiry report options as the request message
- The remaining expiry time from the request message with no adjustment made for the bridge's processing time

If this value is not set, the expiry time is set to *unlimited*.

MQIIH_REPLY_FORMAT_NONE

Sets the MQIIH.Format field of the reply to MQFMT_NONE.

The initial value of this field is MQIIH_NONE.

Format (MQCHAR8)

MQ format name of data that follows MQIIH.

This specifies the MQ format name of the data that follows the MQIIH structure.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *Format* field in MQMD.

The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

LTermOverride (MQCHAR8)

Logical terminal override.

This is placed in the IO PCB field. It is optional; if it is not specified the TPIPE name is used. It is ignored if the first byte is blank, or null.

MQIIH – LTermOverride field

The length of this field is given by MQ_LTERM_OVERRIDE_LENGTH. The initial value of this field is 8 blank characters.

MFSMapName (MQCHAR8)

Message format services map name.

This is placed in the IO PCB field. It is optional. On input it represents the MID, on output it represents the MOD. It is ignored if the first byte is blank or null.

The length of this field is given by MQ_MFS_MAP_NAME_LENGTH. The initial value of this field is 8 blank characters.

ReplyToFormat (MQCHAR8)

MQ format name of reply message.

This is the MQ format name of the reply message that will be sent in response to the current message. The rules for coding this are the same as those for the *Format* field in MQMD.

The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

Reserved (MQCHAR)

Reserved.

This is a reserved field; it must be blank.

SecurityScope (MQCHAR)

Security scope.

This indicates the desired IMS security processing. The following values are defined:

MQISS_CHECK

Check security scope.

An ACEE is built in the control region, but not in the dependent region.

MQISS_FULL

Full security scope.

A cached ACEE is built in the control region and a non-cached ACEE is built in the dependent region. If you use MQISS_FULL, you must ensure that the user ID for which the ACEE is built has access to the resources used in the dependent region.

If neither MQISS_CHECK nor MQISS_FULL is specified for this field, MQISS_CHECK is assumed.

The initial value of this field is MQISS_CHECK.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQIIH_STRUC_ID

Identifier for IMS information header structure.

For the C programming language, the constant `MQIIH_STRUC_ID_ARRAY` is also defined; this has the same value as `MQIIH_STRUC_ID`, but is an array of characters instead of a string.

The initial value of this field is `MQIIH_STRUC_ID`.

StrucLength (MQLONG)

Length of MQIIH structure.

The value must be:

MQIIH_LENGTH_1

Length of IMS information header structure.

The initial value of this field is `MQIIH_LENGTH_1`.

TranInstancId (MQBYTE16)

Transaction instance identifier.

This field is used by output messages from IMS so is ignored on first input. If *TranState* is set to `MQITS_IN_CONVERSATION`, this must be provided in the next input, and all subsequent inputs, to enable IMS to correlate the messages to the correct conversation. The following special value may be used:

MQITII_NONE

No transaction instance id.

For the C programming language, the constant `MQITII_NONE_ARRAY` is also defined; this has the same value as `MQITII_NONE`, but is an array of characters instead of a string.

The length of this field is given by `MQ_TRAN_INSTANCE_ID_LENGTH`. The initial value of this field is `MQITII_NONE`.

TranState (MQCHAR)

Transaction state.

This indicates the IMS conversation state. This is ignored on first input because no conversation exists. On subsequent inputs it indicates whether a conversation is active or not. On output it is set by IMS. The value must be one of the following:

MQITS_IN_CONVERSATION

In conversation.

MQITS_NOT_IN_CONVERSATION

Not in conversation.

MQITS_ARCHITECTED

Return transaction state data in architected form.

This value is used only with the `IMS /DISPLAY TRAN` command. It causes the transaction state data to be returned in the IMS architected form instead of character form. See the *WebSphere MQ Application Programming Guide* for further details.

MQIIH – Version field

The initial value of this field is MQITS_NOT_IN_CONVERSATION.

Version (MQLONG)

Structure version number.

The value must be:

MQIIH_VERSION_1

Version number for IMS information header structure.

The following constant specifies the version number of the current version:

MQIIH_CURRENT_VERSION

Current version of IMS information header structure.

The initial value of this field is MQIIH_VERSION_1.

Initial values and language declarations

Table 50. Initial values of fields in MQIIH

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQIIH_STRUC_ID	'IIHb'
<i>Version</i>	MQIIH_VERSION_1	1
<i>StrucLength</i>	MQIIH_LENGTH_1	84
<i>Encoding</i>	None	0
<i>CodedCharSetId</i>	None	0
<i>Format</i>	MQFMT_NONE	Blanks
<i>Flags</i>	MQIIH_NONE	0
<i>LTermOverride</i>	None	Blanks
<i>MFSMapName</i>	None	Blanks
<i>ReplyToFormat</i>	MQFMT_NONE	Blanks
<i>Authenticator</i>	MQIAUT_NONE	Blanks
<i>TranInstanceId</i>	MQITII_NONE	Nulls
<i>TranState</i>	MQITS_NOT_IN_CONVERSATION	'b'
<i>CommitMode</i>	MQICM_COMMIT_THEN_SEND	'0'
<i>SecurityScope</i>	MQISS_CHECK	'C'
<i>Reserved</i>	None	'b'

Notes:

1. The symbol 'b' represents a single blank character.
2. In the C programming language, the macro variable MQIIH_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQIIH MyIIH = {MQIIH_DEFAULT};
```

C declaration

```

typedef struct tagMQIIH MQIIH;
struct tagMQIIH {
    MQCHAR4  StrucId;           /* Structure identifier */
    MQLONG   Version;          /* Structure version number */
    MQLONG   StrucLength;      /* Length of MQIIH structure */
    MQLONG   Encoding;         /* Reserved */
    MQLONG   CodedCharSetId;   /* Reserved */
    MQCHAR8  Format;           /* MQ format name of data that follows
                               MQIIH */
    MQLONG   Flags;           /* Flags */
    MQCHAR8  LTermOverride;    /* Logical terminal override */
    MQCHAR8  MFSSMapName;      /* Message format services map name */
    MQCHAR8  ReplyToFormat;    /* MQ format name of reply message */
    MQCHAR8  Authenticator;    /* RACF password or passticket */
    MQBYTE16 TranInstanceId;   /* Transaction instance identifier */
    MQCHAR   TranState;        /* Transaction state */
    MQCHAR   CommitMode;       /* Commit mode */
    MQCHAR   SecurityScope;    /* Security scope */
    MQCHAR   Reserved;         /* Reserved */
};

```

COBOL declaration

```

** MQIIH structure
10 MQIIH.
** Structure identifier
15 MQIIH-STRUCID PIC X(4).
** Structure version number
15 MQIIH-VERSION PIC S9(9) BINARY.
** Length of MQIIH structure
15 MQIIH-STRUCLength PIC S9(9) BINARY.
** Reserved
15 MQIIH-ENCODING PIC S9(9) BINARY.
** Reserved
15 MQIIH-CODEDCHARSETID PIC S9(9) BINARY.
** MQ format name of data that follows MQIIH
15 MQIIH-FORMAT PIC X(8).
** Flags
15 MQIIH-FLAGS PIC S9(9) BINARY.
** Logical terminal override
15 MQIIH-LTERMOVERRIDE PIC X(8).
** Message format services map name
15 MQIIH-MFSSMAPNAME PIC X(8).
** MQ format name of reply message
15 MQIIH-REPLYTOFORMAT PIC X(8).
** RACF password or passticket
15 MQIIH-AUTHENTICATOR PIC X(8).
** Transaction instance identifier
15 MQIIH-TRANINSTANCEID PIC X(16).
** Transaction state
15 MQIIH-TRANSTATE PIC X.
** Commit mode
15 MQIIH-COMMITMODE PIC X.
** Security scope
15 MQIIH-SECURITYSCOPE PIC X.
** Reserved
15 MQIIH-RESERVED PIC X.

```

PL/I declaration

```

dcl
1 MQIIH based,
3 StrucId char(4), /* Structure identifier */
3 Version fixed bin(31), /* Structure version number */
3 StrucLength fixed bin(31), /* Length of MQIIH structure */

```

MQIIH – Language declarations

```
3 Encoding      fixed bin(31), /* Reserved */
3 CodedCharSetId fixed bin(31), /* Reserved */
3 Format        char(8),        /* MQ format name of data that follows
                               MQIIH */
3 Flags        fixed bin(31), /* Flags */
3 LTermOverride char(8),        /* Logical terminal override */
3 MFSSMapName  char(8),        /* Message format services map name */
3 ReplyToFormat char(8),        /* MQ format name of reply message */
3 Authenticator char(8),        /* RACF password or passticket */
3 TranInstanceId char(16),      /* Transaction instance identifier */
3 TranState    char(1),        /* Transaction state */
3 CommitMode   char(1),        /* Commit mode */
3 SecurityScope char(1),       /* Security scope */
3 Reserved     char(1);        /* Reserved */
```

System/390 assembler declaration

```
MQIIH          DSECT
MQIIH_STRUCID  DS   CL4   Structure identifier
MQIIH_VERSION  DS   F     Structure version number
MQIIH_STRUCLNGTH DS  F     Length of MQIIH structure
MQIIH_ENCODING DS   F     Reserved
MQIIH_CODEDCCHARSETID DS  F     Reserved
MQIIH_FORMAT   DS   CL8   MQ format name of data that follows
*              MQIIH
MQIIH_FLAGS    DS   F     Flags
MQIIH_LTERMOVERRIDE DS  CL8 Logical terminal override
MQIIH_MFSSMAPNAME DS  CL8 Message format services map name
MQIIH_REPLYTOFORMAT DS  CL8 MQ format name of reply message
MQIIH_AUTHENTICATOR DS  CL8 RACF password or passticket
MQIIH_TRANINSTANCEID DS  XL16 Transaction instance identifier
MQIIH_TRANSTATE DS   CL1  Transaction state
MQIIH_COMMITMODE DS   CL1  Commit mode
MQIIH_SECURITYSCOPE DS  CL1 Security scope
MQIIH_RESERVED DS   CL1  Reserved
*
MQIIH_LENGTH   EQU   *-MQIIH
ORG   MQIIH
MQIIH_AREA     DS   CL(MQIIH_LENGTH)
```

Visual Basic declaration

```
Type MQIIH
  StrucId      As String*4 'Structure identifier'
  Version      As Long     'Structure version number'
  StrucLength  As Long     'Length of MQIIH structure'
  Encoding     As Long     'Reserved'
  CodedCharSetId As Long   'Reserved'
  Format       As String*8 'MQ format name of data that follows MQIIH'
  Flags       As Long     'Flags'
  LTermOverride As String*8 'Logical terminal override'
  MFSSMapName As String*8 'Message format services map name'
  ReplyToFormat As String*8 'MQ format name of reply message'
  Authenticator As String*8 'RACF password or passticket'
  TranInstanceId As MQBYTE16 'Transaction instance identifier'
  TranState     As String*1 'Transaction state'
  CommitMode    As String*1 'Commit mode'
  SecurityScope As String*1 'Security scope'
  Reserved     As String*1 'Reserved'
End Type
```

Chapter 10. MQMD – Message descriptor

The following table summarizes the fields in the structure.

Table 51. Fields in MQMD

Field	Description	Page
<i>StrucId</i>	Structure identifier	194
<i>Version</i>	Structure version number	196
<i>Report</i>	Options for report messages	184
<i>MsgType</i>	Message type	172
<i>Expiry</i>	Message lifetime	151
<i>Feedback</i>	Feedback or reason code	153
<i>Encoding</i>	Numeric encoding of message data	150
<i>CodedCharSetId</i>	Character set identifier of message data	147
<i>Format</i>	Format name of message data	157
<i>Priority</i>	Message priority	176
<i>Persistence</i>	Message persistence	175
<i>MsgId</i>	Message identifier	170
<i>CorrelId</i>	Correlation identifier	149
<i>BackoutCount</i>	Backout counter	147
<i>ReplyToQ</i>	Name of reply queue	182
<i>ReplyToQMgr</i>	Name of reply queue manager	183
<i>UserIdentifier</i>	User identifier	194
<i>AccountingToken</i>	Accounting token	144
<i>ApplIdentityData</i>	Application data relating to identity	146
<i>PutApplType</i>	Type of application that put the message	178
<i>PutApplName</i>	Name of application that put the message	177
<i>PutDate</i>	Date when message was put	181
<i>PutTime</i>	Time when message was put	181
<i>ApplOriginData</i>	Application data relating to origin	146
Note: The remaining fields are ignored if <i>Version</i> is less than MQMD_VERSION_2.		
<i>GroupId</i>	Group identifier	164
<i>MsgSeqNumber</i>	Sequence number of logical message within group	172
<i>Offset</i>	Offset of data in physical message from start of logical message	173
<i>MsgFlags</i>	Message flags	165
<i>OriginalLength</i>	Length of original message	174

Overview

Availability:

- Version 1: All

MQMD – Message descriptor

- Version 2: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems

Purpose: The MQMD structure contains the control information that accompanies the application data when a message travels between the sending and receiving applications. The structure is an input/output parameter on the MQGET, MQPUT, and MQPUT1 calls.

Version: The current version of MQMD is MQMD_VERSION_2, but this version is not supported in all environments (see above). Applications that are intended to be portable between several environments must ensure that the required version of MQMD is supported in all of the environments concerned. Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions that follow.

The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQMD that is supported by the environment, but with the initial value of the *Version* field set to MQMD_VERSION_1. To use fields that are not present in the version-1 structure, the application must set the *Version* field to the version number of the version required.

A declaration for the version-1 structure is available with the name MQMD1.

Character set and encoding: Data in MQMD must be in the character set and encoding of the local queue manager; these are given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE, respectively. However, if the application is running as an MQ client, the structure must be in the character set and encoding of the client.

If the sending and receiving queue managers use different character sets or encodings, the data in MQMD is converted automatically. It is not necessary for the application to convert the MQMD.

Using different versions of MQMD: A version-2 MQMD is generally equivalent to using a version-1 MQMD and prefixing the message data with an MQMDE structure. However, if all of the fields in the MQMDE structure have their default values, the MQMDE can be omitted. A version-1 MQMD plus MQMDE are used as described below.

- On the MQPUT and MQPUT1 calls, if the application provides a version-1 MQMD, the application can optionally prefix the message data with an MQMDE, setting the *Format* field in MQMD to MQFMT_MD_EXTENSION to indicate that an MQMDE is present. If the application does not provide an MQMDE, the queue manager assumes default values for the fields in the MQMDE.

Note: Several of the fields that exist in the version-2 MQMD but not the version-1 MQMD are input/output fields on the MQPUT and MQPUT1 calls. However, the queue manager does *not* return any values in the equivalent fields in the MQMDE on output from the MQPUT and MQPUT1 calls; if the application requires those output values, it must use a version-2 MQMD.

- On the MQGET call, if the application provides a version-1 MQMD, the queue manager prefixes the message returned with an MQMDE, but only if one or

more of the fields in the MQMDE has a non-default value. The *Format* field in MQMD will have the value MQFMT_MD_EXTENSION to indicate that an MQMDE is present.

- On WebSphere MQ for z/OS, the MQMD language declarations provided with Version 5.3 and later have changed as follows:
 - The PL/I and C language declarations contain the additional MQMD version 2 fields and a new MQMD2 structure initialized with version 2 values.
 - You can configure the System/390 Assembler declaration through the DCLVER and VERSION parameters of the CMQMDA macro.
 - The COBOL language declarations contained in CMQMDL and CMQMDV are for a version 1 MQMD, and additional copybooks CMQMD2L and CMQMD2V are provided that contain the MQMD version 2 structure.

For all languages, the default values for MQMD are those for a version 1 structure, and an additional, explicit MQMD1 structure is available (copybooks CMQMD1L and CMQMD1V in COBOL). This structure is provided for applications that have to deal with both version 1 and version 2 MQMDs.

The default values that the queue manager used for the fields in the MQMDE are the same as the initial values of those fields, shown in Table 55 on page 208.

When a message is on a transmission queue, some of the fields in MQMD are set to particular values; see Chapter 25, “MQXQH – Transmission-queue header”, on page 317 for details.

Message context: Certain fields in MQMD contain the message context. There are two types of message context: *identity context* and *origin context*. Usually:

- Identity context relates to the application that *originally* put the message
- Origin context relates to the application that *most-recently* put the message.

These two applications can be the same application, but they can also be different applications (for example, when a message is forwarded from one application to another).

Although identity and origin context usually have the meanings described above, the content of both types of context fields in MQMD actually depends on the MQPMO_*_CONTEXT options that are specified when the message is put. As a result, identity context does not necessarily relate to the application that originally put the message, and origin context does not necessarily relate to the application that most-recently put the message – it depends on the design of the application suite.

There is one class of application that never alters message context, namely the message channel agent (MCA). MCAs that receive messages from remote queue managers use the context option MQPMO_SET_ALL_CONTEXT on the MQPUT or MQPUT1 call. This allows the receiving MCA to preserve exactly the message context that travelled with the message from the sending MCA. However, the result is that the origin context does not relate to the application that most-recently put the message (the receiving MCA), but instead relates to an earlier application that put the message (possibly the originating application itself).

In the descriptions below, the context fields are described as though they are used as described above. For more information about message context, see the *WebSphere MQ Application Programming Guide*.

Fields

The MQMD structure contains the following fields; the fields are described in **alphabetic order**:

AccountingToken (MQBYTE32)

Accounting token.

This is part of the **identity context** of the message. For more information about message context, see “Overview” on page 141; also see the *WebSphere MQ Application Programming Guide*.

AccountingToken allows an application to cause work done as a result of the message to be appropriately charged. The queue manager treats this information as a string of bits and does not check its content.

When the queue manager generates this information, it is set as follows:

- The first byte of the field is set to the length of the accounting information present in the bytes that follow; this length is in the range zero through 30, and is stored in the first byte as a binary integer.
- The second and subsequent bytes (as specified by the length field) are set to the accounting information appropriate to the environment.
 - On z/OS the accounting information is set to:
 - For z/OS batch, the accounting information from the JES JOB card or from a JES ACCT statement in the EXEC card (comma separators are changed to X'FF'). This information is truncated, if necessary, to 31 bytes.
 - For TSO, the user's account number.
 - For CICS, the LU 6.2 unit of work identifier (UEPUOWDS) (26 bytes).
 - For IMS, the 8-character PSB name concatenated with the 16-character IMS recovery token.
 - On OS/400, the accounting information is set to the accounting code for the job.
 - On Compaq OpenVMS Alpha, Compaq NonStop Kernel, and UNIX systems, the accounting information is set to the numeric user identifier, in ASCII characters.
 - On OS/2, PC DOS, Windows 3.1, Windows 95, Windows 98, the accounting information is set to the ASCII character '1'.
 - On Windows, the accounting information is set to a Windows NT security identifier (SID) in a compressed format. The SID uniquely identifies the user identifier stored in the *UserIdentifier* field. When the SID is stored in the *AccountingToken* field, the 6-byte Identifier Authority (located in the third and subsequent bytes of the SID) is omitted. For example, if the Windows NT SID is 28 bytes long, 22 bytes of SID information are stored in the *AccountingToken* field.
- The last byte is set to the accounting-token type, one of the following values:
 - MQACTT_CICS_LUOW_ID**
CICS LUOW identifier.
 - MQACTT_DOS_DEFAULT**
PC DOS default accounting token.
 - MQACTT_NT_SECURITY_ID**
Windows security identifier.
 - MQACTT_OS2_DEFAULT**
OS/2 default accounting token.

MQACTT_OS400_ACCOUNT_TOKEN	OS/400 accounting token.
MQACTT_UNIX_NUMERIC_ID	UNIX systems numeric identifier.
MQACTT_WINDOWS_DEFAULT	Windows 3.1, Windows 95, Windows 98 default accounting token.
MQACTT_USER	User-defined accounting token.
MQACTT_UNKNOWN	Unknown accounting-token type.

The accounting-token type is set to an explicit value only in the following environments: AIX, HP-UX, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems. In other environments, the accounting-token type is set to the value MQACTT_UNKNOWN. In these environments the *PutApplType* field can be used to deduce the type of accounting token received.

- All other bytes are set to binary zero.

On VSE/ESA, this is a reserved field.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. If neither MQPMO_SET_IDENTITY_CONTEXT nor MQPMO_SET_ALL_CONTEXT is specified, this field is ignored on input and is an output-only field. For more information on message context, see the *WebSphere MQ Application Programming Guide*.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *AccountingToken* that was transmitted with the message. If the message has no context, the field is entirely binary zero.

This is an output field for the MQGET call.

This field is not subject to any translation based on the character set of the queue manager—the field is treated as a string of bits, and not as a string of characters.

The queue manager does nothing with the information in this field. The application must interpret the information if it wants to use the information for accounting purposes.

The following special value may be used for the *AccountingToken* field:

MQACT_NONE

No accounting token is specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQACT_NONE_ARRAY is also defined; this has the same value as MQACT_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_ACCOUNTING_TOKEN_LENGTH. The initial value of this field is MQACT_NONE.

MQMD – ApplIdentityData field

ApplIdentityData (MQCHAR32)

Application data relating to identity.

This is part of the **identity context** of the message. For more information about message context, see “Overview” on page 141; also see the *WebSphere MQ Application Programming Guide*.

ApplIdentityData is information that is defined by the application suite, and can be used to provide additional information about the message or its originator. The queue manager treats this information as character data, but does not define the format of it. When the queue manager generates this information, it is entirely blank.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. If a null character is present, the null and any following characters are converted to blanks by the queue manager. If neither MQPMO_SET_IDENTITY_CONTEXT nor MQPMO_SET_ALL_CONTEXT is specified, this field is ignored on input and is an output-only field. For more information on message context, see the *WebSphere MQ Application Programming Guide*.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *ApplIdentityData* that was transmitted with the message. If the message has no context, the field is entirely blank.

On VSE/ESA, this is a reserved field.

This is an output field for the MQGET call. The length of this field is given by MQ_APPL_IDENTITY_DATA_LENGTH. The initial value of this field is the null string in C, and 32 blank characters in other programming languages.

ApplOriginData (MQCHAR4)

Application data relating to origin.

This is part of the **origin context** of the message. For more information about message context, see “Overview” on page 141; also see the *WebSphere MQ Application Programming Guide*.

ApplOriginData is information that is defined by the application suite that can be used to provide additional information about the origin of the message. For example, it could be set by applications running with suitable user authority to indicate whether the identity data is trusted.

The queue manager treats this information as character data, but does not define the format of it. When the queue manager generates this information, it is entirely blank.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. Any information following a null character within the field is discarded. The null character and any following characters are converted to blanks by the queue manager. If MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *ApplOriginData* that was transmitted with the message. If the message has no context, the field is entirely blank.

On VSE/ESA, this is a reserved field.

This is an output field for the MQGET call. The length of this field is given by MQ_APPL_ORIGIN_DATA_LENGTH. The initial value of this field is the null string in C, and 4 blank characters in other programming languages.

BackoutCount (MQLONG)

Backout counter.

This is a count of the number of times the message has been previously returned by the MQGET call as part of a unit of work, and subsequently backed out. It is provided as an aid to the application in detecting processing errors that are based on message content. The count excludes MQGET calls that specified any of the MQGMO_BROWSE_* options.

The accuracy of this count is affected by the *HardenGetBackout* queue attribute; see Chapter 40, “Attributes for queues”, on page 457.

On z/OS, a value of 255 means that the message has been backed out 255 or more times; the value returned is never greater than 255.

On VSE/ESA, this is a reserved field.

This is an output field for the MQGET call. It is ignored for the MQPUT and MQPUT1 calls. The initial value of this field is 0.

CodedCharSetId (MQLONG)

Character set identifier of message data.

This specifies the character set identifier of character data in the message.

Note: Character data in MQMD and the other MQ data structures that are parameters on calls must be in the character set of the queue manager. This is defined by the queue manager’s *CodedCharSetId* attribute; see Chapter 43, “Attributes for the queue manager”, on page 501 for details of this attribute.

The following special values can be used:

MQCCSI_Q_MGR

Queue manager’s character set identifier.

Character data in the message is in the queue manager’s character set.

On the MQPUT and MQPUT1 calls, the queue manager changes this value in the MQMD sent with the message to the true character-set identifier of the queue manager. As a result, the value MQCCSI_Q_MGR is never returned by the MQGET call.

MQCCSI_DEFAULT

Default coded character set identifier.

|
|

MQMD – CodedCharSetId field

The *CodedCharSetId* of the data in the *String* field is defined by the *CodedCharSetId* field in the header structure that precedes the MQCFH structure, or by the *CodedCharSetId* field in the MQMD if the MQCFH is at the start of the message.

MQCCSI_INHERIT

Inherit character-set identifier of this structure.

Character data in the message is in the same character set as this structure; this is the queue-manager's character set. (For MQMD only, MQCCSI_INHERIT has the same meaning as MQCCSI_Q_MGR).

The queue manager changes this value in the MQMD sent with the message to the actual character-set identifier of MQMD. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

MQCCSI_INHERIT cannot be used if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

This value is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

MQCCSI_EMBEDDED

Embedded character set identifier.

Character data in the message is in a character set whose identifier is contained within the message data itself. There can be any number of character-set identifiers embedded within the message data, applying to different parts of the data. This value must be used for PCF messages that contain data in a mixture of character sets. PCF messages have a format name of MQFMT_PCF.

Specify this value only on the MQPUT and MQPUT1 calls. If it is specified on the MQGET call, it prevents conversion of the message.

On the MQPUT and MQPUT1 calls, the queue manager changes the values MQCCSI_Q_MGR and MQCCSI_INHERIT in the MQMD sent with the message as described above, but does not change the MQMD specified on the MQPUT or MQPUT1 call. No other check is carried out on the value specified.

Applications that retrieve messages should compare this field against the value the application is expecting; if the values differ, the application may need to convert character data in the message.

If the MQGMO_CONVERT option is specified on the MQGET call, this field is an input/output field. The value specified by the application is the coded character-set identifier to which the message data should be converted if necessary. If conversion is successful or unnecessary, the value is unchanged (except that the value MQCCSI_Q_MGR or MQCCSI_INHERIT is converted to the actual value). If conversion is unsuccessful, the value after the MQGET call represents the coded character-set identifier of the unconverted message that is returned to the application.

Otherwise, this is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQCCSI_Q_MGR.

CorrelId (MQBYTE24)

Correlation identifier.

This is a byte string that the application can use to relate one message to another, or to relate the message to other work that the application is performing. The correlation identifier is a permanent property of the message, and persists across restarts of the queue manager. Because the correlation identifier is a byte string and not a character string, the correlation identifier is *not* converted between character sets when the message flows from one queue manager to another.

For the MQPUT and MQPUT1 calls, the application can specify any value. The queue manager transmits this value with the message and delivers it to the application that issues the get request for the message.

If the application specifies MQPMO_NEW_CORREL_ID, the queue manager generates a unique correlation identifier³ which is sent with the message, and also returned to the sending application on output from the MQPUT or MQPUT1 call.

When the queue manager or a message channel agent generates a report message, it sets the *CorrelId* field in the way specified by the *Report* field of the original message, either MQRO_COPY_MSG_ID_TO_CORREL_ID or MQRO_PASS_CORREL_ID. Applications which generate report messages should also do this.

For the MQGET call, *CorrelId* is one of the five fields that can be used to select a particular message to be retrieved from the queue. See the description of the *MsgId* field for details of how to specify values for this field.

Specifying MQCI_NONE as the correlation identifier has the same effect as *not* specifying MQMO_MATCH_CORREL_ID, that is, *any* correlation identifier will match.

If the MQGMO_MSG_UNDER_CURSOR option is specified in the *GetMsgOpts* parameter on the MQGET call, this field is ignored.

On return from an MQGET call, the *CorrelId* field is set to the correlation identifier of the message returned (if any).

The following special values may be used:

MQCI_NONE

No correlation identifier is specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQCI_NONE_ARRAY is also defined; this has the same value as MQCI_NONE, but is an array of characters instead of a string.

3. A *CorrelId* generated by the queue manager consists of a 4-byte product identifier ('AMQb' or 'CSQb' in either ASCII or EBCDIC, where 'b' represents a blank), followed by a product-specific implementation of a unique string. In WebSphere MQ this contains the first 12 characters of the queue-manager name, and a value derived from the system clock. All queue managers that can intercommunicate must therefore have names that differ in the first 12 characters, in order to ensure that message identifiers are unique. The ability to generate a unique string also depends upon the system clock not being changed backward. To eliminate the possibility of a message identifier generated by the queue manager duplicating one generated by the application, the application should avoid generating identifiers with initial characters in the range A through I in ASCII or EBCDIC (X'41' through X'49' and X'C1' through X'C9'). However, the application is not prevented from generating identifiers with initial characters in these ranges.

MQMD – CorrelId field

MQCI_NEW_SESSION

Message is the start of a new session.

This value is recognized by the CICS bridge as indicating the start of a new session, that is, the start of a new sequence of messages.

For the C programming language, the constant MQCI_NEW_SESSION_ARRAY is also defined; this has the same value as MQCI_NEW_SESSION, but is an array of characters instead of a string.

For the MQGET call, this is an input/output field. For the MQPUT and MQPUT1 calls, this is an input field if MQPMO_NEW_CORREL_ID is *not* specified, and an output field if MQPMO_NEW_CORREL_ID *is* specified. The length of this field is given by MQ_CORREL_ID_LENGTH. The initial value of this field is MQCI_NONE.

Encoding (MQLONG)

Numeric encoding of message data.

This specifies the numeric encoding of numeric data in the message; it does not apply to numeric data in the MQMD structure itself. The numeric encoding defines the representation used for binary integers, packed-decimal integers, and floating-point numbers.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that the field is valid. The following special value is defined:

MQENC_NATIVE

Native machine encoding.

The encoding is the default for the programming language and machine on which the application is running.

Note: The value of this constant depends on the programming language and environment. For this reason, applications must be compiled using the header, macro, COPY, or INCLUDE files appropriate to the environment in which the application will run.

Applications that put messages should normally specify MQENC_NATIVE. Applications that retrieve messages should compare this field against the value MQENC_NATIVE; if the values differ, the application may need to convert numeric data in the message. The MQGMO_CONVERT option can be used to request the queue manager to convert the message as part of the processing of the MQGET call. See Appendix D, “Machine encodings”, on page 571 for details of how the *Encoding* field is constructed.

If the MQGMO_CONVERT option is specified on the MQGET call, this field is an input/output field. The value specified by the application is the encoding to which the message data should be converted if necessary. If conversion is successful or unnecessary, the value is unchanged. If conversion is unsuccessful, the value after the MQGET call represents the encoding of the unconverted message that is returned to the application.

In other cases, this is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQENC_NATIVE.

Expiry (MQLONG)

Message lifetime.

This is a period of time expressed in tenths of a second, set by the application that puts the message. The message becomes eligible to be discarded if it has not been removed from the destination queue before this period of time elapses.

The value is decremented to reflect the time the message spends on the destination queue, and also on any intermediate transmission queues if the put is to a remote queue. It may also be decremented by message channel agents to reflect transmission times, if these are significant. Likewise, an application forwarding this message to another queue might decrement the value if necessary, if it has retained the message for a significant time. However, the expiration time is treated as approximate, and the value need not be decremented to reflect small time intervals.

When the message is retrieved by an application using the MQGET call, the *Expiry* field represents the amount of the original expiry time that still remains.

After a message's expiry time has elapsed, it becomes eligible to be discarded by the queue manager. In the current implementations, the message is discarded when a browse or nonbrowse MQGET call occurs that would have returned the message had it not already expired. For example, a nonbrowse MQGET call with the *MatchOptions* field in MQGMO set to MQMO_NONE reading from a FIFO ordered queue will cause all the expired messages to be discarded up to the first unexpired message. With a priority ordered queue, the same call will discard expired messages of higher priority and messages of an equal priority that arrived on the queue before the first unexpired message.

A message that has expired is never returned to an application (either by a browse or a non-browse MQGET call), so the value in the *Expiry* field of the message descriptor after a successful MQGET call is either greater than zero, or the special value MQEI_UNLIMITED.

If a message is put on a remote queue, the message may expire (and be discarded) whilst it is on an intermediate transmission queue, before the message reaches the destination queue.

A report is generated when an expired message is discarded, if the message specified one of the MQRO_EXPIRATION_* report options. If none of these options is specified, no such report is generated; the message is assumed to be no longer relevant after this time period (perhaps because a later message has superseded it).

Any other program that discards messages based on expiry time must also send an appropriate report message if one was requested.

Notes:

1. If a message is put with an *Expiry* time of zero, the MQPUT or MQPUT1 call fails with reason code MQRC_EXPIRY_ERROR; no report message is generated in this case.
2. Since a message whose expiry time has elapsed may not actually be discarded until later, there may be messages on a queue that have passed their expiry

MQMD – Expiry field

time, and which are not therefore eligible for retrieval. These messages nevertheless count towards the number of messages on the queue for all purposes, including depth triggering.

3. An expiration report is generated, if requested, when the message is actually discarded, not when it becomes eligible for discarding.
4. Discarding of an expired message, and the generation of an expiration report if requested, are never part of the application's unit of work, even if the message was scheduled for discarding as a result of an MQGET call operating within a unit of work.
5. If a nearly-expired message is retrieved by an MQGET call within a unit of work, and the unit of work is subsequently backed out, the message may become eligible to be discarded before it can be retrieved again.
6. If a nearly-expired message is locked by an MQGET call with MQGMO_LOCK, the message may become eligible to be discarded before it can be retrieved by an MQGET call with MQGMO_MSG_UNDER_CURSOR; reason code MQRC_NO_MSG_UNDER_CURSOR is returned on this subsequent MQGET call if that happens.
7. When a request message with an expiry time greater than zero is retrieved, the application can take one of the following actions when it sends the reply message:
 - Copy the remaining expiry time from the request message to the reply message.
 - Set the expiry time in the reply message to an explicit value greater than zero.
 - Set the expiry time in the reply message to MQEI_UNLIMITED.

The action to take depends on the design of the application suite. However, the default action for putting messages to a dead-letter (undelivered-message) queue should be to preserve the remaining expiry time of the message, and to continue to decrement it.

8. Trigger messages are always generated with MQEI_UNLIMITED.
9. A message (normally on a transmission queue) which has a *Format* name of MQFMT_XMIT_Q_HEADER has a second message descriptor within the MQXQH. It therefore has two *Expiry* fields associated with it. The following additional points should be noted in this case:
 - When an application puts a message on a remote queue, the queue manager places the message initially on a local transmission queue, and prefixes the application message data with an MQXQH structure. The queue manager sets the values of the two *Expiry* fields to be the same as that specified by the application.

If an application puts a message directly on a local transmission queue, the message data must already begin with an MQXQH structure, and the format name must be MQFMT_XMIT_Q_HEADER (but the queue manager does not enforce this). In this case the application need not set the values of these two *Expiry* fields to be the same. (The queue manager does not check that the *Expiry* field within the MQXQH contains a valid value, or even that the message data is long enough to include it.)

- When a message with a *Format* name of MQFMT_XMIT_Q_HEADER is retrieved from a queue (whether this is a normal or a transmission queue), the queue manager decrements *both* these *Expiry* fields with the time spent waiting on the queue. No error is raised if the message data is not long enough to include the *Expiry* field in the MQXQH.

- The queue manager uses the *Expiry* field in the separate message descriptor (that is, not the one in the message descriptor embedded within the MQXQH structure) to test whether the message is eligible for discarding.
- If the initial values of the two *Expiry* fields were different, it is therefore possible for the *Expiry* time in the separate message descriptor when the message is retrieved to be greater than zero (so the message is not eligible for discarding), while the time according to the *Expiry* field in the MQXQH has elapsed. In this case the *Expiry* field in the MQXQH is set to zero.

The following special value is recognized:

MQEI_UNLIMITED

Unlimited lifetime.

The message has an unlimited expiration time.

On VSE/ESA, the value of *Expiry* must be MQEI_UNLIMITED.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQEI_UNLIMITED.

Expired messages on z/OS

On WebSphere MQ for z/OS, messages that have expired are discarded by the next appropriate MQGET call. However, if no such call occurs, the expired message is not discarded, and, for some queues, a large number of expired messages can accumulate. To remedy this, you can set the queue manager to scan queues periodically and discard expired messages on one or more queues.

There are two ways of doing this:

Periodic scan

You can specify a period using the EXPRYINT (expiry interval) queue manager attribute. Each time the expiry interval is reached, the queue manager looks for candidate queues that are worth scanning to discard expired messages.

The queue manager maintains information about the expired messages on each queue, and knows whether a scan for expired messages is worthwhile. So, only a selection of queues is scanned at any time.

Shared queues are scanned only by one queue manager in a queue-sharing group. Generally, it is the first queue manager to restart, or the first to have EXPRYINT set. If this queue manager terminates, another queue manager in the queue-sharing group takes over the queue scanning. Set the expiry interval value for all queue managers within a queue-sharing group to the same value.

Explicit request

Issue the REFRESH QMGR TYPE(EXPIRY) command, specifying the queue or queues that you want scanned.

Feedback (MQLONG)

Feedback or reason code.

This is used with a message of type MQMT_REPORT to indicate the nature of the report, and is only meaningful with that type of message. The field can contain one of the MQFB_* values, or one of the MQRC_* values. Feedback codes are grouped as follows:

MQMD – Feedback field

MQFB_NONE

No feedback provided.

MQFB_SYSTEM_FIRST

Lowest value for system-generated feedback.

MQFB_SYSTEM_LAST

Highest value for system-generated feedback.

The range of system-generated feedback codes MQFB_SYSTEM_FIRST through MQFB_SYSTEM_LAST includes the general feedback codes listed below (MQFB_*), and also the reason codes (MQRC_*) that can occur when the message cannot be put on the destination queue.

MQFB_APPL_FIRST

Lowest value for application-generated feedback.

MQFB_APPL_LAST

Highest value for application-generated feedback.

Applications that generate report messages should not use feedback codes in the system range (other than MQFB_QUIT), unless they wish to simulate report messages generated by the queue manager or message channel agent.

On the MQPUT or MQPUT1 calls, the value specified must either be MQFB_NONE, or be within the system range or application range. This is checked whatever the value of *MsgType*.

General feedback codes:

MQFB_COA

Confirmation of arrival on the destination queue (see MQRO_COA).

MQFB_COD

Confirmation of delivery to the receiving application (see MQRO_COD).

MQFB_EXPIRATION

Message expired.

Message was discarded because it had not been removed from the destination queue before its expiry time had elapsed.

MQFB_PAN

Positive action notification (see MQRO_PAN).

MQFB_NAN

Negative action notification (see MQRO_NAN).

MQFB_QUIT

Application should end.

This can be used by a workload scheduling program to control the number of instances of an application program that are running. Sending an MQMT_REPORT message with this feedback code to an instance of the application program indicates to that instance that it should stop processing. However, adherence to this convention is a matter for the application; it is not enforced by the queue manager.

IMS-bridge feedback codes:

Note: The IMS bridge is not available when using the reduced function form of WebSphere MQ for z/OS supplied with WebSphere Application Server.

When the IMS bridge receives a nonzero IMS-OTMA sense code, the IMS bridge converts the sense code from hexadecimal to decimal, adds the value MQFB_IMS_ERROR (300), and places the result in the *Feedback* field of the reply message. This results in the feedback code having a value in the range MQFB_IMS_FIRST (301) through MQFB_IMS_LAST (399) when an IMS-OTMA error has occurred.

The following feedback codes can be generated by the IMS bridge:

MQFB_DATA_LENGTH_ZERO

Data length zero.

A segment length was zero in the application data of the message.

MQFB_DATA_LENGTH_NEGATIVE

Data length negative.

A segment length was negative in the application data of the message.

MQFB_DATA_LENGTH_TOO_BIG

Data length too big.

A segment length was too big in the application data of the message.

MQFB_BUFFER_OVERFLOW

Buffer overflow.

The value of one of the length fields would cause the data to overflow the message buffer.

MQFB_LENGTH_OFF_BY_ONE

Length in error by one.

The value of one of the length fields was one byte too short.

MQFB_IIH_ERROR

MQIIH structure not valid or missing.

The *Format* field in MQMD specifies MQFMT_IMS, but the message does not begin with a valid MQIIH structure.

MQFB_NOT_AUTHORIZED_FOR_IMS

Userid not authorized for use in IMS.

The user ID contained in the message descriptor MQMD, or the password contained in the *Authenticator* field in the MQIIH structure, failed the validation performed by the IMS bridge. As a result the message was not passed to IMS.

MQFB_IMS_ERROR

Unexpected error returned by IMS.

An unexpected error was returned by IMS. Consult the MQSeries error log on the system on which the IMS bridge resides for more information about the error.

MQFB_IMS_FIRST

Lowest value for IMS-generated feedback.

IMS-generated feedback codes occupy the range MQFB_IMS_FIRST (300) through MQFB_IMS_LAST (399). The IMS-OTMA sense code itself is *Feedback* minus MQFB_IMS_ERROR.

MQFB_IMS_LAST

Highest value for IMS-generated feedback.

MQMD – Feedback field

CICS-bridge feedback codes:

Note: The CICS bridge is not available when using the reduced function form of WebSphere MQ for z/OS supplied with WebSphere Application Server. The following feedback codes can be generated by the CICS bridge:

MQFB_CICS_APPL_ABENDED

Application abended.

The application program specified in the message abended. This feedback code occurs only in the *Reason* field of the MQDLH structure.

MQFB_CICS_APPL_NOT_STARTED

Application cannot be started.

The EXEC CICS LINK for the application program specified in the message failed. This feedback code occurs only in the *Reason* field of the MQDLH structure.

MQFB_CICS_BRIDGE_FAILURE

CICS bridge terminated abnormally without completing normal error processing.

MQFB_CICS_CCSID_ERROR

Character set identifier not valid.

MQFB_CICS_CIH_ERROR

CICS information header structure missing or not valid.

MQFB_CICS_COMMAREA_ERROR

Length of CICS commarea not valid.

MQFB_CICS_CORREL_ID_ERROR

Correlation identifier not valid.

MQFB_CICS_DLQ_ERROR

Dead-letter queue not available.

The CICS bridge task was unable to copy a reply to this request to the dead-letter queue. The request was backed out.

MQFB_CICS_ENCODING_ERROR

Encoding not valid.

MQFB_CICS_INTERNAL_ERROR

CICS bridge encountered an unexpected error.

This feedback code occurs only in the *Reason* field of the MQDLH structure.

MQFB_CICS_NOT_AUTHORIZED

User identifier not authorized or password not valid.

This feedback code occurs only in the *Reason* field of the MQDLH structure.

MQFB_CICS_UOW_BACKED_OUT

Unit of work backed out.

The unit of work was backed out, for one of the following reasons:

- A failure was detected while processing another request within the same unit of work.
- A CICS abend occurred while the unit of work was in progress.

MQFB_CICS_UOW_ERROR

Unit-of-work control field *UOWControl* not valid.

MQ reason codes: For exception report messages, *Feedback* contains an MQ reason code. Among possible reason codes are:

MQRC_PUT_INHIBITED

(2051, X'803') Put calls inhibited for the queue.

MQRC_Q_FULL

(2053, X'805') Queue already contains maximum number of messages.

MQRC_NOT_AUTHORIZED

(2035, X'7F3') Not authorized for access.

MQRC_Q_SPACE_NOT_AVAILABLE

(2056, X'808') No space available on disk for queue.

MQRC_PERSISTENT_NOT_ALLOWED

(2048, X'800') Queue does not support persistent messages.

MQRC_MSG_TOO_BIG_FOR_Q_MGR

(2031, X'7EF') Message length greater than maximum for queue manager.

MQRC_MSG_TOO_BIG_FOR_Q

(2030, X'7EE') Message length greater than maximum for queue.

For a full list of reason codes, see “Reason codes” on page 528.

This is an output field for the MQGET call, and an input field for MQPUT and MQPUT1 calls. The initial value of this field is MQFB_NONE.

Format (MQCHAR8)

Format name of message data.

This is a name that the sender of the message may use to indicate to the receiver the nature of the data in the message. Any characters that are in the queue manager’s character set may be specified for the name, but it is recommended that the name be restricted to the following:

- Uppercase A through Z
- Numeric digits 0 through 9

If other characters are used, it may not be possible to translate the name between the character sets of the sending and receiving queue managers.

The name should be padded with blanks to the length of the field, or a null character used to terminate the name before the end of the field; the null and any subsequent characters are treated as blanks. Do not specify a name with leading or embedded blanks. For the MQGET call, the queue manager returns the name padded with blanks to the length of the field.

The queue manager does not check that the name complies with the recommendations described above.

Names beginning “MQ” in upper, lower, and mixed case have meanings that are defined by the queue manager; you should not use names beginning with these letters for your own formats. The queue manager built-in formats are:

MQFMT_NONE

No format name.

MQMD – Format field

The nature of the data is undefined. This means that the data cannot be converted when the message is retrieved from a queue using the MQGMO_CONVERT option.

If MQGMO_CONVERT is specified on the MQGET call, and the character set or encoding of data in the message differs from that specified in the *MsgDesc* parameter, the message is returned with the following completion and reason codes (assuming no other errors):

- Completion code MQCC_WARNING and reason code MQRC_FORMAT_ERROR if the MQFMT_NONE data is at the beginning of the message.
- Completion code MQCC_OK and reason code MQRC_NONE if the MQFMT_NONE data is at the end of the message (that is, preceded by one or more MQ header structures). The MQ header structures are converted to the requested character set and encoding in this case.

For the C programming language, the constant MQFMT_NONE_ARRAY is also defined; this has the same value as MQFMT_NONE, but is an array of characters instead of a string.

MQFMT_ADMIN

Command server request/reply message.

The message is a command-server request or reply message in programmable command format (PCF). Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call. Refer to the *WebSphere MQ Programmable Command Formats and Administration Interface* book for more information about using programmable command format messages.

For the C programming language, the constant MQFMT_ADMIN_ARRAY is also defined; this has the same value as MQFMT_ADMIN, but is an array of characters instead of a string.

MQFMT_CICS

CICS information header.

Note: CICS connection is not available when using the reduced function form of WebSphere MQ for z/OS supplied with WebSphere Application Server.

The message data begins with the CICS information header MQCIH, which is followed by the application data. The format name of the application data is given by the *Format* field in the MQCIH structure.

On z/OS, the MQGMO_CONVERT option can be specified on the MQGET call to convert messages that have format MQFMT_CICS.

For the C programming language, the constant MQFMT_CICS_ARRAY is also defined; this has the same value as MQFMT_CICS, but is an array of characters instead of a string.

MQFMT_COMMAND_1

Type 1 command reply message.

The message is an MQSC command-server reply message containing the object count, completion code, and reason code. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant `MQFMT_COMMAND_1_ARRAY` is also defined; this has the same value as `MQFMT_COMMAND_1`, but is an array of characters instead of a string.

MQFMT_COMMAND_2

Type 2 command reply message.

The message is an MQSC command-server reply message containing information about the object(s) requested. Messages of this format can be converted if the `MQGMO_CONVERT` option is specified on the `MQGET` call.

For the C programming language, the constant `MQFMT_COMMAND_2_ARRAY` is also defined; this has the same value as `MQFMT_COMMAND_2`, but is an array of characters instead of a string.

MQFMT_DEAD_LETTER_HEADER

Dead-letter header.

The message data begins with the dead-letter header `MQDLH`. The data from the original message immediately follows the `MQDLH` structure. The format name of the original message data is given by the *Format* field in the `MQDLH` structure; see Chapter 7, “`MQDLH` – Dead-letter header”, on page 83 for details of this structure. Messages of this format can be converted if the `MQGMO_CONVERT` option is specified on the `MQGET` call.

COA and COD reports are not generated for messages which have a *Format* of `MQFMT_DEAD_LETTER_HEADER`.

For the C programming language, the constant `MQFMT_DEAD_LETTER_HEADER_ARRAY` is also defined; this has the same value as `MQFMT_DEAD_LETTER_HEADER`, but is an array of characters instead of a string.

MQFMT_DIST_HEADER

Distribution-list header.

The message data begins with the distribution-list header `MQDH`; this includes the arrays of `MQOR` and `MQPMR` records. The distribution-list header may be followed by additional data. The format of the additional data (if any) is given by the *Format* field in the `MQDH` structure; see Chapter 6, “`MQDH` – Distribution header”, on page 75 for details of this structure. Messages with format `MQFMT_DIST_HEADER` can be converted if the `MQGMO_CONVERT` option is specified on the `MQGET` call.

This format is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

For the C programming language, the constant `MQFMT_DIST_HEADER_ARRAY` is also defined; this has the same value as `MQFMT_DIST_HEADER`, but is an array of characters instead of a string.

MQFMT_EVENT

Event message.

MQMD – Format field

Note: Events are not available when using the reduced function form of WebSphere MQ for z/OS supplied with WebSphere Application Server.

The message is an MQ event message that reports an event that occurred. Event messages have the same structure as programmable commands; Refer to the *WebSphere MQ Programmable Command Formats and Administration Interface* book for more information about this structure, and to the *WebSphere MQ Event Monitoring* book for information about events.

Version-1 event messages can be converted in all environments if the MQGMO_CONVERT option is specified on the MQGET call. Version-2 event messages can be converted only on z/OS.

For the C programming language, the constant MQFMT_EVENT_ARRAY is also defined; this has the same value as MQFMT_EVENT, but is an array of characters instead of a string.

MQFMT_IMS

IMS information header.

Note: IMS connection is not available when using the reduced function form of WebSphere MQ for z/OS supplied with WebSphere Application Server.

The message data begins with the IMS information header MQIIH, which is followed by the application data. The format name of the application data is given by the *Format* field in the MQIIH structure.

In the following environments, the MQGMO_CONVERT option can be specified on the MQGET call to convert messages that have format MQFMT_IMS: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

For the C programming language, the constant MQFMT_IMS_ARRAY is also defined; this has the same value as MQFMT_IMS, but is an array of characters instead of a string.

MQFMT_IMS_VAR_STRING

IMS variable string.

Note: IMS connection is not available when using the reduced function form of WebSphere MQ for z/OS supplied with WebSphere Application Server.

The message is an IMS variable string, which is a string of the form 11zzccc, where:

11 is a 2-byte length field specifying the total length of the IMS variable string item. This length is equal to the length of 11 (2 bytes), plus the length of zz (2 bytes), plus the length of the character string itself. 11 is a 2-byte binary integer in the encoding specified by the *Encoding* field.

zz is a 2-byte field containing flags that are significant to IMS. zz is a

byte string consisting of two MQBYTE fields, and is transmitted without change from sender to receiver (that is, zz is not subject to any conversion).

ccc is a variable-length character string containing 11-4 characters. ccc is in the character set specified by the *CodedCharSetId* field.

On z/OS, it is valid for the message data to consist of a sequence of IMS variable strings butted together, with each string being of the form 11zzccc. Note that there must be no bytes skipped between successive IMS variable strings. This means that if the first string has an odd length, the second string will be misaligned, that is, it will not begin on a boundary that is a multiple of two. Care must be taken constructing such strings on machines that require alignment of elementary data types.

In the following environments, the MQGMO_CONVERT option can be specified on the MQGET call to convert messages that have format MQFMT_IMS_VAR_STRING: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

For the C programming language, the constant MQFMT_IMS_VAR_STRING_ARRAY is also defined; this has the same value as MQFMT_IMS_VAR_STRING, but is an array of characters instead of a string.

MQFMT_MD_EXTENSION

Message-descriptor extension.

The message data begins with the message-descriptor extension MQMDE, and is optionally followed by other data (usually the application message data). The format name, character set, and encoding of the data which follows the MQMDE is given by the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQMDE. See Chapter 11, “MQMDE – Message descriptor extension”, on page 203 for details of this structure. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

This format is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

For the C programming language, the constant MQFMT_MD_EXTENSION_ARRAY is also defined; this has the same value as MQFMT_MD_EXTENSION, but is an array of characters instead of a string.

MQFMT_PCF

User-defined message in programmable command format (PCF).

The message is a user-defined message that conforms to the structure of a programmable command format (PCF) message. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call. Refer to the *WebSphere MQ Programmable Command Formats and Administration Interface* book for more information about using programmable command format messages.

MQMD – Format field

For the C programming language, the constant `MQFMT_PCF_ARRAY` is also defined; this has the same value as `MQFMT_PCF`, but is an array of characters instead of a string.

MQFMT_REF_MSG_HEADER

Reference message header.

The message data begins with the reference message header `MQRMH`, and is optionally followed by other data. The format name, character set, and encoding of the data is given by the *Format*, *CodedCharSetId*, and *Encoding* fields in the `MQRMH`. See Chapter 18, “`MQRMH` – Reference message header”, on page 271 for details of this structure. Messages of this format can be converted if the `MQGMO_CONVERT` option is specified on the `MQGET` call.

This format is supported in the following environments: AIX, HP-UX, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

For the C programming language, the constant `MQFMT_REF_MSG_HEADER_ARRAY` is also defined; this has the same value as `MQFMT_REF_MSG_HEADER`, but is an array of characters instead of a string.

MQFMT_RF_HEADER

Rules and formatting header.

The message data begins with the rules and formatting header `MQRFH`, and is optionally followed by other data. The format name, character set, and encoding of the data (if any) is given by the *Format*, *CodedCharSetId*, and *Encoding* fields in the `MQRFH`. Messages of this format can be converted if the `MQGMO_CONVERT` option is specified on the `MQGET` call.

This format is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

For the C programming language, the constant `MQFMT_RF_HEADER_ARRAY` is also defined; this has the same value as `MQFMT_RF_HEADER`, but is an array of characters instead of a string.

MQFMT_RF_HEADER_2

Rules and formatting header version 2.

The message data begins with the version-2 rules and formatting header `MQRFH2`, and is optionally followed by other data. The format name, character set, and encoding of the optional data (if any) is given by the *Format*, *CodedCharSetId*, and *Encoding* fields in the `MQRFH2`. Messages of this format can be converted if the `MQGMO_CONVERT` option is specified on the `MQGET` call.

This format is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

For the C programming language, the constant MQFMT_RF_HEADER_2_ARRAY is also defined; this has the same value as MQFMT_RF_HEADER_2, but is an array of characters instead of a string.

MQFMT_STRING

Message consisting entirely of characters.

The application message data can be either an SBCS string (single-byte character set), or a DBCS string (double-byte character set). Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_STRING_ARRAY is also defined; this has the same value as MQFMT_STRING, but is an array of characters instead of a string.

MQFMT_TRIGGER

Trigger message.

The message is a trigger message, described by the MQTM structure; see Chapter 21, “MQTM – Trigger message”, on page 291 for details of this structure. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_TRIGGER_ARRAY is also defined; this has the same value as MQFMT_TRIGGER, but is an array of characters instead of a string.

MQFMT_WORK_INFO_HEADER

Work information header.

The message data begins with the work information header MQWIH, which is followed by the application data. The format name of the application data is given by the *Format* field in the MQWIH structure.

On z/OS, the MQGMO_CONVERT option can be specified on the MQGET call to convert the *user data* in messages that have format MQFMT_WORK_INFO_HEADER. However, the MQWIH structure itself is always returned in the queue-manager’s character set and encoding (that is, the MQWIH structure is converted whether or not the MQGMO_CONVERT option is specified).

For the C programming language, the constant MQFMT_WORK_INFO_HEADER_ARRAY is also defined; this has the same value as MQFMT_WORK_INFO_HEADER, but is an array of characters instead of a string.

MQFMT_XMIT_Q_HEADER

Transmission queue header.

The message data begins with the transmission queue header MQXQH. The data from the original message immediately follows the MQXQH structure. The format name of the original message data is given by the *Format* field in the MQMD structure which is part of the transmission queue header MQXQH. See Chapter 25, “MQXQH – Transmission-queue header”, on page 317 for details of this structure.

COA and COD reports are not generated for messages which have a *Format* of MQFMT_XMIT_Q_HEADER.

MQMD – Format field

For the C programming language, the constant `MQFMT_XMIT_Q_HEADER_ARRAY` is also defined; this has the same value as `MQFMT_XMIT_Q_HEADER`, but is an array of characters instead of a string.

This is an output field for the `MQGET` call, and an input field for the `MQPUT` and `MQPUT1` calls. The length of this field is given by `MQ_FORMAT_LENGTH`. The initial value of this field is `MQFMT_NONE`.

GroupId (MQBYTE24)

Group identifier.

This is a byte string that is used to identify the particular message group or logical message to which the physical message belongs. *GroupId* is also used if segmentation is allowed for the message. In all of these cases, *GroupId* has a non-null value, and one or more of the following flags is set in the *MsgFlags* field:

- MQMF_MSG_IN_GROUP
- MQMF_LAST_MSG_IN_GROUP
- MQMF_SEGMENT
- MQMF_LAST_SEGMENT
- MQMF_SEGMENTATION_ALLOWED

If none of these flags is set, *GroupId* has the special null value `MQGI_NONE`.

This field need not be set by the application on the `MQPUT` or `MQGET` call if:

- On the `MQPUT` call, `MQPMO_LOGICAL_ORDER` is specified.
- On the `MQGET` call, `MQMO_MATCH_GROUP_ID` is *not* specified.

These are the recommended ways of using these calls for messages that are not report messages. However, if the application requires more control, or the call is `MQPUT1`, the application must ensure that *GroupId* is set to an appropriate value.

Message groups and segments can be processed correctly only if the group identifier is unique. For this reason, *applications should not generate their own group identifiers*; instead, applications should do one of the following:

- If `MQPMO_LOGICAL_ORDER` is specified, the queue manager automatically generates a unique group identifier for the first message in the group or segment of the logical message, and uses that group identifier for the remaining messages in the group or segments of the logical message, so the application does not need to take any special action. This is the recommended procedure.
- If `MQPMO_LOGICAL_ORDER` is *not* specified, the application should request the queue manager to generate the group identifier, by setting *GroupId* to `MQGI_NONE` on the first `MQPUT` or `MQPUT1` call for a message in the group or segment of the logical message. The group identifier returned by the queue manager on output from that call should then be used for the remaining messages in the group or segments of the logical message. If a message group contains segmented messages, the same group identifier must be used for all segments and messages in the group.

When `MQPMO_LOGICAL_ORDER` is not specified, messages in groups and segments of logical messages can be put in any order (for example, in reverse order), but the group identifier must be allocated by the *first* `MQPUT` or `MQPUT1` call that is issued for any of those messages.

On input to the `MQPUT` and `MQPUT1` calls, the queue manager uses the value detailed in Table 61 on page 235. On output from the `MQPUT` and `MQPUT1` calls, the queue manager sets this field to the value that was sent with the message if the

object opened is a single queue and not a distribution list, but leaves it unchanged if the object opened is a distribution list. In the latter case, if the application needs to know the group identifiers generated, the application must provide MQPMR records containing the *GroupId* field.

On input to the MQGET call, the queue manager uses the value detailed in Table 46 on page 116. On output from the MQGET call, the queue manager sets this field to the value for the message retrieved.

The following special value is defined:

MQGI_NONE

No group identifier specified.

The value is binary zero for the length of the field. This is the value that is used for messages that are not in groups, not segments of logical messages, and for which segmentation is not allowed.

For the C programming language, the constant MQGI_NONE_ARRAY is also defined; this has the same value as MQGI_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_GROUP_ID_LENGTH. The initial value of this field is MQGI_NONE. This field is ignored if *Version* is less than MQMD_VERSION_2.

MsgFlags (MQLONG)

Message flags.

These are flags that specify attributes of the message, or control its processing. The flags are divided into the following categories:

- Segmentation flag
- Status flags

These are described in turn.

Segmentation flags: When a message is too big for a queue, an attempt to put the message on the queue usually fails. Segmentation is a technique whereby the queue manager or application splits the message into smaller pieces called segments, and places each segment on the queue as a separate physical message. The application which retrieves the message can either retrieve the segments one by one, or request the queue manager to reassemble the segments into a single message which is returned by the MQGET call. The latter is achieved by specifying the MQGMO_COMPLETE_MSG option on the MQGET call, and supplying a buffer that is big enough to accommodate the complete message. (See Chapter 8, “MQGMO – Get-message options”, on page 95 for details of the MQGMO_COMPLETE_MSG option.) Segmentation of a message can occur at the sending queue manager, at an intermediate queue manager, or at the destination queue manager.

You can specify one of the following to control the segmentation of a message:

MQMF_SEGMENTATION_INHIBITED

Segmentation inhibited.

This option prevents the message being broken into segments by the queue manager. If specified for a message that is already a segment, this option prevents the segment being broken into smaller segments.

MQMD – MsgFlags field

The value of this flag is binary zero. This is the default.

MQMF_SEGMENTATION_ALLOWED

Segmentation allowed.

This option allows the message to be broken into segments by the queue manager. If specified for a message that is already a segment, this option allows the segment to be broken into smaller segments.

MQMF_SEGMENTATION_ALLOWED can be set without either MQMF_SEGMENT or MQMF_LAST_SEGMENT being set.

- On z/OS, the queue manager does not support the segmentation of messages. If a message is too big for the queue, the MQPUT or MQPUT1 call fails with reason code MQRC_MSG_TOO_BIG_FOR_Q. However, the MQMF_SEGMENTATION_ALLOWED option can still be specified, and allows the message to be segmented at a remote queue manager.

When the queue manager segments a message, the queue manager turns on the MQMF_SEGMENT flag in the copy of the MQMD that is sent with each segment, but does not alter the settings of these flags in the MQMD provided by the application on the MQPUT or MQPUT1 call. For the last segment in the logical message, the queue manager also turns on the MQMF_LAST_SEGMENT flag in the MQMD that is sent with the segment.

Note: Care is needed when messages are put with MQMF_SEGMENTATION_ALLOWED but without MQPMO_LOGICAL_ORDER. If the message is:

- Not a segment, and
- Not in a group, and
- Not being forwarded,

the application must remember to reset the *GroupId* field to MQGI_NONE prior to *each* MQPUT or MQPUT1 call, in order to cause a unique group identifier to be generated by the queue manager for each message. If this is not done, unrelated messages could inadvertently end up with the same group identifier, which might lead to incorrect processing subsequently. See the descriptions of the *GroupId* field and the MQPMO_LOGICAL_ORDER option for more information about when the *GroupId* field must be reset.

The queue manager splits messages into segments as necessary in order to ensure that the segments (plus any header data that may be required) fit on the queue. However, there is a lower limit for the size of a segment generated by the queue manager (see below), and only the last segment created from a message can be smaller than this limit. (The lower limit for the size of an application-generated segment is one byte.) Segments generated by the queue manager may be of unequal length. The queue-manager processes the message as follows:

- User-defined formats are split on boundaries which are multiples of 16 bytes. This means that the queue manager will not generate segments that are smaller than 16 bytes (other than the last segment).
- Built-in formats other than MQFMT_STRING are split at points appropriate to the nature of the data present. However, the queue manager never splits a message in the middle of an MQ header structure. This means that a segment containing a single MQ header structure cannot be split further by the queue manager, and as a result the minimum possible segment size for that message is greater than 16 bytes.

The second or later segment generated by the queue manager will begin with one of the following:

- An MQ header structure
- The start of the application message data
- Part-way through the application message data
- MQFMT_STRING is split without regard for the nature of the data present (SBCS, DBCS, or mixed SBCS/DBCS). When the string is DBCS or mixed SBCS/DBCS, this may result in segments which cannot be converted from one character set to another (see below). The queue manager never splits MQFMT_STRING messages into segments that are smaller than 16 bytes (other than the last segment).
- The *Format*, *CodedCharSetId*, and *Encoding* fields in the MQMD of each segment are set by the queue manager to describe correctly the data present at the *start* of the segment; the format name will be either the name of a built-in format, or the name of a user-defined format.
- The *Report* field in the MQMD of segments with *Offset* greater than zero are modified as follows:
 - For each report type, if the report option is MQRO_*_WITH_DATA, but the segment cannot possibly contain any of the first 100 bytes of user data (that is, the data following any MQ header structures that may be present), the report option is changed to MQRO_*.

The queue manager follows the above rules, but otherwise splits messages as it thinks fit; no assumptions should be made about the way that the queue manager will choose to split a particular message.

For *persistent* messages, the queue manager can perform segmentation only within a unit of work:

- If the MQPUT or MQPUT1 call is operating within a user-defined unit of work, that unit of work is used. If the call fails partway through the segmentation process, the queue manager removes any segments that were placed on the queue as a result of the failing call. However, the failure does not prevent the unit of work being committed successfully.
- If the call is operating outside a user-defined unit of work, and there is no user-defined unit of work in existence, the queue manager creates a unit of work just for the duration of the call. If the call is successful, the queue manager commits the unit of work automatically (the application does not need to do this). If the call fails, the queue manager backs out the unit of work.
- If the call is operating outside a user-defined unit of work, but a user-defined unit of work *does* exist, the queue manager is unable to perform segmentation. If the message does not require segmentation, the call can still succeed. But if the message *does* require segmentation, the call fails with reason code MQRC_UOW_NOT_AVAILABLE.

For *nonpersistent* messages, the queue manager does not require a unit of work to be available in order to perform segmentation.

Special consideration must be given to data conversion of messages which may be segmented:

- If data conversion is performed only by the receiving application on the MQGET call, and the application specifies the MQGMO_COMPLETE_MSG option, the data-conversion exit will be

MQMD – MsgFlags field

passed the complete message for the exit to convert, and the fact that the message was segmented will not be apparent to the exit.

- If the receiving application retrieves one segment at a time, the data-conversion exit will be invoked to convert one segment at a time. The exit must therefore be capable of converting the data in a segment independently of the data in any of the other segments.

If the nature of the data in the message is such that arbitrary segmentation of the data on 16-byte boundaries may result in segments which cannot be converted by the exit, or the format is MQFMT_STRING and the character set is DBCS or mixed SBCS/DBCS, the sending application should itself create and put the segments, specifying MQMF_SEGMENTATION_INHIBITED to suppress further segmentation. In this way, the sending application can ensure that each segment contains sufficient information to allow the data-conversion exit to convert the segment successfully.

- If sender conversion is specified for a sending message channel agent (MCA), the MCA converts only messages which are not segments of logical messages; the MCA never attempts to convert messages which are segments.

This flag is an input flag on the MQPUT and MQPUT1 calls, and an output flag on the MQGET call. On the latter call, the queue manager also echoes the value of the flag to the *Segmentation* field in MQGMO.

The initial value of this flag is MQMF_SEGMENTATION_INHIBITED.

Status flags: These are flags that indicate whether the physical message belongs to a message group, is a segment of a logical message, both, or neither. One or more of the following can be specified on the MQPUT or MQPUT1 call, or returned by the MQGET call:

MQMF_MSG_IN_GROUP

Message is a member of a group.

MQMF_LAST_MSG_IN_GROUP

Message is the last logical message in a group.

If this flag is set, the queue manager turns on MQMF_MSG_IN_GROUP in the copy of MQMD that is sent with the message, but does not alter the settings of these flags in the MQMD provided by the application on the MQPUT or MQPUT1 call.

It is valid for a group to consist of only one logical message. If this is the case, MQMF_LAST_MSG_IN_GROUP is set, but the *MsgSeqNumber* field has the value one.

MQMF_SEGMENT

Message is a segment of a logical message.

When MQMF_SEGMENT is specified without MQMF_LAST_SEGMENT, the length of the application message data in the segment (*excluding* the lengths of any MQ header structures that may be present) must be at least one. If the length is zero, the MQPUT or MQPUT1 call fails with reason code MQRC_SEGMENT_LENGTH_ZERO.

- On z/OS, this option is not supported if the message is being put on a queue that has an index type of MQIT_GROUP_ID.

MQMF_LAST_SEGMENT

Message is the last segment of a logical message.

If this flag is set, the queue manager turns on MQMF_SEGMENT in the copy of MQMD that is sent with the message, but does not alter the settings of these flags in the MQMD provided by the application on the MQPUT or MQPUT1 call.

It is valid for a logical message to consist of only one segment. If this is the case, MQMF_LAST_SEGMENT is set, but the *Offset* field has the value zero.

When MQMF_LAST_SEGMENT is specified, it is permissible for the length of the application message data in the segment (*excluding* the lengths of any header structures that may be present) to be zero.

- On z/OS, this option is not supported if the message is being put on a queue that has an index type of MQIT_GROUP_ID.

The application must ensure that these flags are set correctly when putting messages. If MQPMO_LOGICAL_ORDER is specified, or was specified on the preceding MQPUT call for the queue handle, the settings of the flags must be consistent with the group and segment information retained by the queue manager for the queue handle. The following conditions apply to *successive* MQPUT calls for the queue handle when MQPMO_LOGICAL_ORDER is specified:

- If there is no current group or logical message, all of these flags (and combinations of them) are valid.
- Once MQMF_MSG_IN_GROUP has been specified, it must remain on until MQMF_LAST_MSG_IN_GROUP is specified. The call fails with reason code MQRC_INCOMPLETE_GROUP if this condition is not satisfied.
- Once MQMF_SEGMENT has been specified, it must remain on until MQMF_LAST_SEGMENT is specified. The call fails with reason code MQRC_INCOMPLETE_MSG if this condition is not satisfied.
- Once MQMF_SEGMENT has been specified without MQMF_MSG_IN_GROUP, MQMF_MSG_IN_GROUP must remain *off* until after MQMF_LAST_SEGMENT has been specified. The call fails with reason code MQRC_INCOMPLETE_MSG if this condition is not satisfied.

Table 61 on page 235 shows the valid combinations of the flags, and the values used for various fields.

These flags are input flags on the MQPUT and MQPUT1 calls, and output flags on the MQGET call. On the latter call, the queue manager also echoes the values of the flags to the *GroupStatus* and *SegmentStatus* fields in MQGMO.

Default flags: The following can be specified to indicate that the message has default attributes:

MQMF_NONE

No message flags (default message attributes).

This inhibits segmentation, and indicates that the message is not in a group and is not a segment of a logical message. MQMF_NONE is defined to aid program documentation. It is not intended that this flag be used with any other, but as its value is zero, such use cannot be detected.

The *MsgFlags* field is partitioned into subfields; for details see Appendix E, “Report options and message flags”, on page 575.

MQMD – MsgFlags field

The initial value of this field is MQMF_NONE. This field is ignored if *Version* is less than MQMD_VERSION_2.

MsgId (MQBYTE24)

Message identifier.

This is a byte string that is used to distinguish one message from another. Generally, no two messages should have the same message identifier, although this is not disallowed by the queue manager. The message identifier is a permanent property of the message, and persists across restarts of the queue manager. Because the message identifier is a byte string and not a character string, the message identifier is *not* converted between character sets when the message flows from one queue manager to another.

For the MQPUT and MQPUT1 calls, if MQMI_NONE or MQPMO_NEW_MSG_ID is specified by the application, the queue manager generates a unique message identifier⁴ when the message is put, and places it in the message descriptor sent with the message. The queue manager also returns this message identifier in the message descriptor belonging to the sending application. The application can use this value to record information about particular messages, and to respond to queries from other parts of the application.

If the message is being put to a distribution list, the queue manager generates unique message identifiers as necessary, but the value of the *MsgId* field in MQMD is unchanged on return from the call, even if MQMI_NONE or MQPMO_NEW_MSG_ID was specified. If the application needs to know the message identifiers generated by the queue manager, the application must provide MQPMR records containing the *MsgId* field.

The sending application can also specify a particular value for the message identifier, other than MQMI_NONE; this stops the queue manager generating a unique message identifier. An application that is forwarding a message can use this facility to propagate the message identifier of the original message.

The queue manager does not itself make any use of this field except to:

- Generate a unique value if requested, as described above
- Deliver the value to the application that issues the get request for the message
- Copy the value to the *CorrelId* field of any report message that it generates about this message (depending on the *Report* options)

When the queue manager or a message channel agent generates a report message, it sets the *MsgId* field in the way specified by the *Report* field of the original message, either MQRO_NEW_MSG_ID or MQRO_PASS_MSG_ID. Applications that generate report messages should also do this.

4. A *MsgId* generated by the queue manager consists of a 4-byte product identifier ('AMQb' or 'CSQb' in either ASCII or EBCDIC, where 'b' represents a blank), followed by a product-specific implementation of a unique string. In WebSphere MQ this contains the first 12 characters of the queue-manager name, and a value derived from the system clock. All queue managers that can intercommunicate must therefore have names that differ in the first 12 characters, in order to ensure that message identifiers are unique. The ability to generate a unique string also depends upon the system clock not being changed backward. To eliminate the possibility of a message identifier generated by the queue manager duplicating one generated by the application, the application should avoid generating identifiers with initial characters in the range A through I in ASCII or EBCDIC (X'41' through X'49' and X'C1' through X'C9'). However, the application is not prevented from generating identifiers with initial characters in these ranges.

For the MQGET call, *MsgId* is one of the five fields that can be used to select a particular message to be retrieved from the queue. Normally the MQGET call returns the next message on the queue, but if a particular message is required, this can be obtained by specifying one or more of the five selection criteria, in any combination; these fields are:

MsgId
CorrelId
GroupId
MsgSeqNumber
Offset

The application sets one or more of these field to the values required, and then sets the corresponding MQMO_* match options in the *MatchOptions* field in MQGMO to indicate that those fields should be used as selection criteria. Only messages that have the specified values in those fields are candidates for retrieval. The default for the *MatchOptions* field (if not altered by the application) is to match both the message identifier and the correlation identifier.

- On z/OS, the selection criteria that can be used may be restricted by the type of index used for the queue. See the *IndexType* queue attribute for further details.

Normally, the message returned is the *first* message on the queue that satisfies the selection criteria. But if MQGMO_BROWSE_NEXT is specified, the message returned is the *next* message that satisfies the selection criteria; the scan for this message starts with the message *following* the current cursor position.

Note: The queue is scanned sequentially for a message that satisfies the selection criteria, so retrieval times will be slower than if no selection criteria are specified, especially if many messages have to be scanned before a suitable one is found.

See Table 46 on page 116 for more information about how selection criteria are used in various situations.

Specifying MQMI_NONE as the message identifier has the same effect as *not* specifying MQMO_MATCH_MSG_ID, that is, *any* message identifier will match.

This field is ignored if the MQGMO_MSG_UNDER_CURSOR option is specified in the *GetMsgOpts* parameter on the MQGET call.

On return from an MQGET call, the *MsgId* field is set to the message identifier of the message returned (if any).

The following special value may be used:

MQMI_NONE

No message identifier is specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQMI_NONE_ARRAY is also defined; this has the same value as MQMI_NONE, but is an array of characters instead of a string.

This is an input/output field for the MQGET, MQPUT, and MQPUT1 calls. The length of this field is given by MQ_MSG_ID_LENGTH. The initial value of this field is MQMI_NONE.

MQMD – MsgSeqNumber field

MsgSeqNumber (MQLONG)

Sequence number of logical message within group.

Sequence numbers start at 1, and increase by 1 for each new logical message in the group, up to a maximum of 999 999 999. A physical message which is not in a group has a sequence number of 1.

This field need not be set by the application on the MQPUT or MQGET call if:

- On the MQPUT call, MQPMO_LOGICAL_ORDER is specified.
- On the MQGET call, MQMO_MATCH_MSG_SEQ_NUMBER is *not* specified.

These are the recommended ways of using these calls for messages that are not report messages. However, if the application requires more control, or the call is MQPUT1, the application must ensure that *MsgSeqNumber* is set to an appropriate value.

On input to the MQPUT and MQPUT1 calls, the queue manager uses the value detailed in Table 61 on page 235. On output from the MQPUT and MQPUT1 calls, the queue manager sets this field to the value that was sent with the message.

On input to the MQGET call, the queue manager uses the value detailed in Table 46 on page 116. On output from the MQGET call, the queue manager sets this field to the value for the message retrieved.

The initial value of this field is one. This field is ignored if *Version* is less than MQMD_VERSION_2.

MsgType (MQLONG)

Message type.

This indicates the type of the message. Message types are grouped as follows:

MQMT_SYSTEM_FIRST

Lowest value for system-defined message types.

MQMT_SYSTEM_LAST

Highest value for system-defined message types.

The following values are currently defined within the system range:

MQMT_DATAGRAM

Message not requiring a reply.

The message is one that does not require a reply.

MQMT_REQUEST

Message requiring a reply.

The message is one that requires a reply.

The name of the queue to which the reply should be sent must be specified in the *ReplyToQ* field. The *Report* field indicates how the *MsgId* and *CorrelId* of the reply are to be set.

MQMT_REPLY

Reply to an earlier request message.

The message is the reply to an earlier request message (MQMT_REQUEST). The message should be sent to the queue indicated by

the *ReplyToQ* field of the request message. The *Report* field of the request should be used to control how the *MsgId* and *CorrelId* of the reply are set.

Note: The queue manager does not enforce the request-reply relationship; this is an application responsibility.

MQMT_REPORT

Report message.

The message is reporting on some expected or unexpected occurrence, usually related to some other message (for example, a request message was received which contained data that was not valid). The message should be sent to the queue indicated by the *ReplyToQ* field of the message descriptor of the original message. The *Feedback* field should be set to indicate the nature of the report. The *Report* field of the original message can be used to control how the *MsgId* and *CorrelId* of the report message should be set.

Report messages generated by the queue manager or message channel agent are always sent to the *ReplyToQ* queue, with the *Feedback* and *CorrelId* fields set as described above.

Other values within the system range may be defined in future versions of the MQI, and are accepted by the MQPUT and MQPUT1 calls without error.

Application-defined values can also be used. They must be within the following range:

MQMT_APPL_FIRST

Lowest value for application-defined message types.

MQMT_APPL_LAST

Highest value for application-defined message types.

For the MQPUT and MQPUT1 calls, the *MsgType* value must be within either the system-defined range or the application-defined range; if it is not, the call fails with reason code MQRC_MSG_TYPE_ERROR.

This is an output field for the MQGET call, and an input field for MQPUT and MQPUT1 calls. The initial value of this field is MQMT_DATAGRAM.

Offset (MQLONG)

Offset of data in physical message from start of logical message.

This is the offset in bytes of the data in the physical message from the start of the logical message of which the data forms part. This data is called a *segment*. The offset is in the range 0 through 999 999 999. A physical message which is not a segment of a logical message has an offset of zero.

This field need not be set by the application on the MQPUT or MQGET call if:

- On the MQPUT call, MQPMO_LOGICAL_ORDER is specified.
- On the MQGET call, MQMO_MATCH_OFFSET is *not* specified.

These are the recommended ways of using these calls for messages that are not report messages. However, if the application does not comply with these conditions, or the call is MQPUT1, the application must ensure that *Offset* is set to an appropriate value.

MQMD – Offset field

On input to the MQPUT and MQPUT1 calls, the queue manager uses the value detailed in Table 61 on page 235. On output from the MQPUT and MQPUT1 calls, the queue manager sets this field to the value that was sent with the message.

For a report message reporting on a segment of a logical message, the *OriginalLength* field (provided it is not MQOL_UNDEFINED) is used to update the offset in the segment information retained by the queue manager.

On input to the MQGET call, the queue manager uses the value detailed in Table 46 on page 116. On output from the MQGET call, the queue manager sets this field to the value for the message retrieved.

The initial value of this field is zero. This field is ignored if *Version* is less than MQMD_VERSION_2.

OriginalLength (MQLONG)

Length of original message.

This field is of relevance only for report messages that are segments. It specifies the length of the message segment to which the report message relates; it does not specify the length of the logical message of which the segment forms part, nor the length of the data in the report message.

Note: When generating a report message for a message that is a segment, the queue manager and message channel agent copy into the MQMD for the report message the *GroupId*, *MsgSeqNumber*, *Offset*, and *MsgFlags*, fields from the original message. As a result, the report message is also a segment. Applications that generate report messages are recommended to do the same, and to ensure that the *OriginalLength* field is set correctly.

The following special value is defined:

MQOL_UNDEFINED

Original length of message not defined.

OriginalLength is an input field on the MQPUT and MQPUT1 calls, but the value provided by the application is accepted only in particular circumstances:

- If the message being put is a segment and is also a report message, the queue manager accepts the value specified. The value must be:
 - Greater than zero if the segment is not the last segment
 - Not less than zero if the segment is the last segment
 - Not less than the length of data present in the message

If these conditions are not satisfied, the call fails with reason code MQRC_ORIGINAL_LENGTH_ERROR.

- If the message being put is a segment but not a report message, the queue manager ignores the field and uses the length of the application message data instead.
- In all other cases, the queue manager ignores the field and uses the value MQOL_UNDEFINED instead.

This is an output field on the MQGET call.

The initial value of this field is MQOL_UNDEFINED. This field is ignored if *Version* is less than MQMD_VERSION_2.

Persistence (MQLONG)

Message persistence.

This indicates whether the message survives system failures and restarts of the queue manager. For the MQPUT and MQPUT1 calls, the value must be one of the following:

MQPER_PERSISTENT

Message is persistent.

This means that the message survives system failures and restarts of the queue manager. Once the message has been put, and the putter's unit of work committed (if the message is put as part of a unit of work), the message is preserved on auxiliary storage. It remains there until the message is removed from the queue, and the getter's unit of work committed (if the message is retrieved as part of a unit of work).

When a persistent message is sent to a remote queue, a store-and-forward mechanism is used to hold the message at each queue manager along the route to the destination, until the message is known to have arrived at the next queue manager.

Persistent messages cannot be placed on:

- Temporary dynamic queues
- shared queues that map to a CFSTRUCT object at CFLEVEL(2) or below, or where the CFSTRUCT object is defined as RECOVER(NO).

Persistent messages can be placed on permanent dynamic queues, and predefined queues.

MQPER_NOT_PERSISTENT

Message is not persistent.

This means that the message does not normally survive system failures or restarts of the queue manager. This applies even if an intact copy of the message is found on auxiliary storage during restart of the queue manager.

In the special case of shared queues, nonpersistent messages *do* survive restarts of queue managers in the queue-sharing group, but do not survive failures of the coupling facility used to store messages on the shared queues.

- On VSE/ESA, MQPER_NOT_PERSISTENT is not supported.

MQPER_PERSISTENCE_AS_Q_DEF

Message has default persistence.

- If the queue is a cluster queue, the persistence of the message is taken from the *DefPersistence* attribute defined at the *destination* queue manager that owns the particular instance of the queue on which the message is placed. Usually, all of the instances of a cluster queue have the same value for the *DefPersistence* attribute, although this is not mandated.

The value of *DefPersistence* is copied into the *Persistence* field when the message is placed on the destination queue. If *DefPersistence* is changed subsequently, messages that have already been placed on the queue are not affected.

- If the queue is not a cluster queue, the persistence of the message is taken from the *DefPersistence* attribute defined at the *local* queue manager, even if the destination queue manager is remote.

MQMD – Persistence field

If there is more than one definition in the queue-name resolution path, the default persistence is taken from the value of this attribute in the *first* definition in the path. This could be:

- An alias queue
- A local queue
- A local definition of a remote queue
- A queue-manager alias
- A transmission queue (for example, the *DefXmitQName* queue)

The value of *DefPersistence* is copied into the *Persistence* field when the message is put. If *DefPersistence* is changed subsequently, messages that have already been put are not affected.

On VSE/ESA, this value is not supported.

Both persistent and nonpersistent messages can exist on the same queue.

When replying to a message, applications should normally use for the reply message the persistence of the request message.

For an MQGET call, the value returned is either MQPER_PERSISTENT or MQPER_NOT_PERSISTENT.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQPER_PERSISTENCE_AS_Q_DEF.

Priority (MQLONG)

Message priority.

For the MQPUT and MQPUT1 calls, the value must be greater than or equal to zero; zero is the lowest priority. The following special value can also be used:

MQPRI_PRIORITY_AS_Q_DEF

Default priority for queue.

- If the queue is a cluster queue, the priority for the message is taken from the *DefPriority* attribute as defined at the *destination* queue manager that owns the particular instance of the queue on which the message is placed. Usually, all of the instances of a cluster queue have the same value for the *DefPriority* attribute, although this is not mandated.

The value of *DefPriority* is copied into the *Priority* field when the message is placed on the destination queue. If *DefPriority* is changed subsequently, messages that have already been placed on the queue are not affected.

- If the queue is not a cluster queue, the priority for the message is taken from the *DefPriority* attribute as defined at the *local* queue manager, even if the destination queue manager is remote.

If there is more than one definition in the queue-name resolution path, the default priority is taken from the value of this attribute in the *first* definition in the path. This could be:

- An alias queue
- A local queue
- A local definition of a remote queue
- A queue-manager alias
- A transmission queue (for example, the *DefXmitQName* queue)

The value of *DefPriority* is copied into the *Priority* field when the message is put. If *DefPriority* is changed subsequently, messages that have already been put are not affected.

The value returned by the MQGET call is always greater than or equal to zero; the value MQPRI_PRIORITY_AS_Q_DEF is never returned.

If a message is put with a priority greater than the maximum supported by the local queue manager (this maximum is given by the *MaxPriority* queue-manager attribute), the message is accepted by the queue manager, but placed on the queue at the queue manager's maximum priority; the MQPUT or MQPUT1 call completes with MQCC_WARNING and reason code MQRC_PRIORITY_EXCEEDS_MAXIMUM. However, the *Priority* field retains the value specified by the application which put the message.

- On z/OS, if a message with a *MsgSeqNumber* of 1 is put to a queue that has a message delivery sequence of MQMDS_PRIORITY and an index type of MQIT_GROUP_ID, the queue may treat the message with a different priority. If the message was placed on the queue with a priority of 0 or 1, it will be processed as though it has a priority of 2. This is because the order of messages placed on this type of queue is optimized to enable efficient group completeness tests. For more information on the message delivery sequence MQMDS_PRIORITY and the index type MQIT_GROUP_ID, see “MsgDeliverySequence (MQLONG)” on page 475.

When replying to a message, applications should normally use for the reply message the priority of the request message. In other situations, specifying MQPRI_PRIORITY_AS_Q_DEF allows priority tuning to be carried out without changing the application.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQPRI_PRIORITY_AS_Q_DEF.

PutAppName (MQCHAR28)

Name of application that put the message.

This is part of the **origin context** of the message. For more information about message context, see “Overview” on page 141; also see the *WebSphere MQ Application Programming Guide*.

The format of the *PutAppName* depends on the value of *PutApplType*.

When this field is set by the queue manager (that is, for all options except MQPMO_SET_ALL_CONTEXT), it is set to value which is determined by the environment:

- On z/OS, the queue manager uses:
 - For z/OS batch, the 8-character job name from the JES JOB card
 - For TSO, the 7-character TSO user identifier
 - For CICS, the 8-character applid, followed by the 4-character tranid
 - For IMS, the 8-character IMS system identifier, followed by the 8-character PSB name
 - For XCF, the 8-character XCF group name, followed by the 16-character XCF member name
 - For a message generated by a queue manager, the first 28 characters of the queue manager name

MQMD – PutAppName field

- For distributed queuing without CICS, the 8-character jobname of the channel initiator followed by the 8-character name of the module putting to the dead-letter queue followed by an 8-character task identifier.
- For MQSeries Java™ language bindings processing with MQSeries for OS/390, the 8-character jobname of the address space created for the OpenEdition® environment. Typically, this will be a TSO user identifier with a single numeric character appended.

The name or names are each padded to the right with blanks, as is any space in the remainder of the field. Where there is more than one name, there is no separator between them.

- On OS/2, PC DOS, and Windows systems, the queue manager uses:
 - For a CICS application, the CICS transaction name
 - For a non-CICS application, the rightmost 28 characters of the fully-qualified name of the executable
- On OS/400, the queue manager uses the fully-qualified job name.
- On Compaq OpenVMS Alpha and Compaq NonStop Kernel, the queue manager uses: the rightmost 28 characters of the fully-qualified name of the executable, if this is available to the queue manager, and blanks otherwise
- On UNIX systems, the queue manager uses:
 - For a CICS application, the CICS transaction name
 - For a non-CICS application, the rightmost 14 characters of the fully-qualified name of the executable if this is available to the queue manager, and blanks otherwise (for example, on AIX)
- On VSE/ESA, the queue manager uses the 8-character applid, followed by the 4-character tranid.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. Any information following a null character within the field is discarded. The null character and any following characters are converted to blanks by the queue manager. If MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *PutAppName* that was transmitted with the message. If the message has no context, the field is entirely blank.

This is an output field for the MQGET call. The length of this field is given by MQ_PUT_APPL_NAME_LENGTH. The initial value of this field is the null string in C, and 28 blank characters in other programming languages.

PutAppType (MQLONG)

Type of application that put the message.

This is part of the **origin context** of the message. For more information about message context, see “Overview” on page 141; also see the *WebSphere MQ Application Programming Guide*.

PutAppType may have one of the following standard types. User-defined types can also be used but should be restricted to values in the range MQAT_USER_FIRST through MQAT_USER_LAST.

MQAT_AIX

AIX application (same value as MQAT_UNIX).

MQAT_BROKER	Broker.
MQAT_CICS	CICS transaction.
MQAT_CICS_BRIDGE	CICS bridge.
MQAT_CICS_VSE	CICS/VSE transaction.
MQAT_DOS	WebSphere MQ client application on PC DOS.
MQAT_DQM	Distributed queue manager agent.
MQAT_GUARDIAN	Tandem Guardian application (same value as MQAT_NSK).
MQAT_IMS	IMS application.
MQAT_IMS_BRIDGE	IMS bridge.
MQAT_JAVA	Java.
MQAT_MVS	MVS or TSO application (same value as MQAT_ZOS).
MQAT_NOTES_AGENT	Lotus Notes [®] Agent application.
MQAT_NSK	Compaq NonStop Kernel application.
MQAT_OS2	OS/2 or Presentation Manager application.
MQAT_OS390	OS/390 application (same value as MQAT_ZOS).
MQAT_OS400	OS/400 application.
MQAT_QMGR	Queue manager.
MQAT_UNIX	UNIX application.
MQAT_VMS	Digital OpenVMS application.
MQAT_VOS	Stratus VOS application.
MQAT_WINDOWS	16-bit Windows application.
MQAT_WINDOWS_NT	32-bit Windows application.

MQMD – PutApplType field

| MQAT_WLM
| z/OS workload manager application.

MQAT_XCF
XCF.

| MQAT_ZOS
| z/OS application.

MQAT_DEFAULT
Default application type.

This is the default application type for the platform on which the application is running.

Note: The value of this constant is environment-specific. Because of this, the application must be compiled using the header, include, or COPY files that are appropriate to the platform on which the application will run.

MQAT_UNKNOWN
Unknown application type.

This value can be used to indicate that the application type is unknown, even though other context information is present.

MQAT_USER_FIRST
Lowest value for user-defined application type.

MQAT_USER_LAST
Highest value for user-defined application type.

The following special value can also occur:

MQAT_NO_CONTEXT
No context information present in message.

This value is set by the queue manager when a message is put with no context (that is, the MQPMO_NO_CONTEXT context option is specified).

When a message is retrieved, *PutApplType* can be tested for this value to decide whether the message has context (it is recommended that *PutApplType* is never set to MQAT_NO_CONTEXT, by an application using MQPMO_SET_ALL_CONTEXT, if any of the other context fields are nonblank).

When the queue manager generates this information as a result of an application put, the field is set to a value that is determined by the environment. Note that on OS/400, it is set to MQAT_OS400; the queue manager never uses MQAT_CICS on OS/400.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. If MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *PutApplType* that was transmitted with the message. If the message has no context, the field is set to MQAT_NO_CONTEXT.

This is an output field for the MQGET call. The initial value of this field is MQAT_NO_CONTEXT.

PutDate (MQCHAR8)

Date when message was put.

This is part of the **origin context** of the message. For more information about message context, see “Overview” on page 141; also see the *WebSphere MQ Application Programming Guide*.

The format used for the date when this field is generated by the queue manager is:

YYYYMMDD

where the characters represent:

YYYY year (four numeric digits)

MM month of year (01 through 12)

DD day of month (01 through 31)

Greenwich Mean Time (GMT) is used for the *PutDate* and *PutTime* fields, subject to the system clock being set accurately to GMT.

On OS/2, the queue manager uses the TZ environment variable to calculate GMT. For more information on setting this variable, refer to the *WebSphere MQ System Administration Guide* book.

If the message was put as part of a unit of work, the date is that when the message was put, and not the date when the unit of work was committed.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. The contents of the field are not checked by the queue manager, except that any information following a null character within the field is discarded. The null character and any following characters are converted to blanks by the queue manager. If MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *PutDate* that was transmitted with the message. If the message has no context, the field is entirely blank.

On VSE/ESA, this is a reserved field.

This is an output field for the MQGET call. The length of this field is given by MQ_PUT_DATE_LENGTH. The initial value of this field is the null string in C, and 8 blank characters in other programming languages.

PutTime (MQCHAR8)

Time when message was put.

This is part of the **origin context** of the message. For more information about message context, see “Overview” on page 141; also see the *WebSphere MQ Application Programming Guide*.

The format used for the time when this field is generated by the queue manager is:

HHMMSSTH

MQMD – PutTime field

where the characters represent (in order):

HH hours (00 through 23)
MM minutes (00 through 59)
SS seconds (00 through 59; see note below)
T tenths of a second (0 through 9)
H hundredths of a second (0 through 9)

Note: If the system clock is synchronized to a very accurate time standard, it is possible on rare occasions for 60 or 61 to be returned for the seconds in *PutTime*. This happens when leap seconds are inserted into the global time standard.

Greenwich Mean Time (GMT) is used for the *PutDate* and *PutTime* fields, subject to the system clock being set accurately to GMT.

On OS/2, the queue manager uses the TZ environment variable to calculate GMT.

If the message was put as part of a unit of work, the time is that when the message was put, and not the time when the unit of work was committed.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. The contents of the field are not checked by the queue manager, except that any information following a null character within the field is discarded. The null character and any following characters are converted to blanks by the queue manager. If MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *PutTime* that was transmitted with the message. If the message has no context, the field is entirely blank.

On VSE/ESA, this is a reserved field.

This is an output field for the MQGET call. The length of this field is given by MQ_PUT_TIME_LENGTH. The initial value of this field is the null string in C, and 8 blank characters in other programming languages.

ReplyToQ (MQCHAR48)

Name of reply queue.

This is the name of the message queue to which the application that issued the get request for the message should send MQMT_REPLY and MQMT_REPORT messages. The name is the local name of a queue that is defined on the queue manager identified by *ReplyToQMgr*. This queue should not be a model queue, although the sending queue manager does not verify this when the message is put.

For the MQPUT and MQPUT1 calls, this field must not be blank if the *MsgType* field has the value MQMT_REQUEST, or if any report messages are requested by the *Report* field. However, the value specified (or substituted; see below) is passed on to the application that issues the get request for the message, whatever the message type.

If the *ReplyToQMgr* field is blank, the local queue manager looks up the *ReplyToQ* name in its own queue definitions. If a local definition of a remote queue exists

with this name, the *ReplyToQ* value in the transmitted message is replaced by the value of the *RemoteQName* attribute from the definition of the remote queue, and this value will be returned in the message descriptor when the receiving application issues an MQGET call for the message. If a local definition of a remote queue does not exist, *ReplyToQ* is unchanged.

If the name is specified, it may contain trailing blanks; the first null character and characters following it are treated as blanks. Otherwise, however, no check is made that the name satisfies the naming rules for queues; this is also true for the name transmitted, if the *ReplyToQ* is replaced in the transmitted message. The only check made is that a name has been specified, if the circumstances require it.

If a reply-to queue is not required, it is recommended (although this is not checked) that the *ReplyToQ* field should be set to blanks, or (in the C programming language) to the null string, or to one or more blanks followed by a null character; the field should not be left uninitialized.

For the MQGET call, the queue manager always returns the name padded with blanks to the length of the field.

If a message that requires a report message cannot be delivered, and the report message also cannot be delivered to the queue specified, both the original message and the report message go to the dead-letter (undelivered-message) queue (see the *DeadLetterQName* attribute described in Chapter 43, “Attributes for the queue manager”, on page 501).

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ReplyToQMgr (MQCHAR48)

Name of reply queue manager.

This is the name of the queue manager to which the reply message or report message should be sent. *ReplyToQ* is the local name of a queue that is defined on this queue manager.

If the *ReplyToQMgr* field is blank, the local queue manager looks up the *ReplyToQ* name in its queue definitions. If a local definition of a remote queue exists with this name, the *ReplyToQMgr* value in the transmitted message is replaced by the value of the *RemoteQMgrName* attribute from the definition of the remote queue, and this value will be returned in the message descriptor when the receiving application issues an MQGET call for the message. If a local definition of a remote queue does not exist, the *ReplyToQMgr* that is transmitted with the message is the name of the local queue manager.

If the name is specified, it may contain trailing blanks; the first null character and characters following it are treated as blanks. Otherwise, however, no check is made that the name satisfies the naming rules for queue managers, or that this name is known to the sending queue manager; this is also true for the name transmitted, if the *ReplyToQMgr* is replaced in the transmitted message. For more information about names, see the *WebSphere MQ Application Programming Guide*.

MQMD – ReplyToQMgr field

If a reply-to queue is not required, it is recommended (although this is not checked) that the *ReplyToQMgr* field should be set to blanks, or (in the C programming language) to the null string, or to one or more blanks followed by a null character; the field should not be left uninitialized.

For the MQGET call, the queue manager always returns the name padded with blanks to the length of the field.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The length of this field is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

Report (MQLONG)

Options for report messages.

A report message is a message about another message, used to inform an application about expected or unexpected events that relate to the original message. The *Report* field enables the application sending the original message to specify which report messages are required, whether the application message data is to be included in them, and also (for both reports and replies) how the message and correlation identifiers in the report or reply message are to be set. Any or all (or none) of the following types of report message can be requested:

- Exception
- Expiration
- Confirm on arrival (COA)
- Confirm on delivery (COD)
- Positive action notification (PAN)
- Negative action notification (NAN)

If more than one type of report message is required, or other report options are needed, the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

The application that receives the report message can determine the reason the report was generated by examining the *Feedback* field in the MQMD; see the *Feedback* field for more details.

Exception options: You can specify one of the options listed below to request an exception report message.

- On VSE/ESA, these options are not supported.

MQRO_EXCEPTION

Exception reports required.

This type of report can be generated by a message channel agent when a message is sent to another queue manager and the message cannot be delivered to the specified destination queue. For example, the destination queue or an intermediate transmission queue might be full, or the message might be too big for the queue.

Generation of the exception report message depends on the persistence of the original message, and the speed of the message channel (normal or fast) through which the original message travels:

- For all persistent messages, and for nonpersistent messages traveling through normal message channels, the exception report is generated *only* if the action specified by the sending application for the error condition can be completed successfully. The sending application can specify one of the following actions to control the disposition of the original message when the error condition arises:
 - MQRO_DEAD_LETTER_Q (this causes the original message to be placed on the dead-letter queue).
 - MQRO_DISCARD_MSG (this causes the original message to be discarded).

If the action specified by the sending application cannot be completed successfully, the original message is left on the transmission queue, and no exception report message is generated.

- For nonpersistent messages traveling through fast message channels, the original message is removed from the transmission queue and the exception report generated *even if* the specified action for the error condition cannot be completed successfully. For example, if MQRO_DEAD_LETTER_Q is specified, but the original message cannot be placed on the dead-letter queue because (say) that queue is full, the exception report message is generated and the original message discarded.

Refer to the *WebSphere MQ Intercommunication* book for more information about normal and fast message channels.

An exception report is not generated if the application that put the original message can be notified synchronously of the problem by means of the reason code returned by the MQPUT or MQPUT1 call.

Applications can also send exception reports, to indicate that a message that it has received cannot be processed (for example, because it is a debit transaction that would cause the account to exceed its credit limit).

Message data from the original message is not included with the report message.

Do not specify more than one of MQRO_EXCEPTION, MQRO_EXCEPTION_WITH_DATA, and MQRO_EXCEPTION_WITH_FULL_DATA.

MQRO_EXCEPTION_WITH_DATA

Exception reports with data required.

This is the same as MQRO_EXCEPTION, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of MQRO_EXCEPTION, MQRO_EXCEPTION_WITH_DATA, and MQRO_EXCEPTION_WITH_FULL_DATA.

MQRO_EXCEPTION_WITH_FULL_DATA

Exception reports with full data required.

This is the same as MQRO_EXCEPTION, except that all of the application message data from the original message is included in the report message.

MQMD – Report field

Do not specify more than one of MQRO_EXCEPTION, MQRO_EXCEPTION_WITH_DATA, and MQRO_EXCEPTION_WITH_FULL_DATA.

Expiration options: You can specify one of the options listed below to request an expiration report message.

- On VSE/ESA, these options are not supported.

MQRO_EXPIRATION

Expiration reports required.

This type of report is generated by the queue manager if the message is discarded prior to delivery to an application because its expiry time has passed (see the *Expiry* field). If this option is not set, no report message is generated if a message is discarded for this reason (even if one of the MQRO_EXCEPTION_* options is specified).

Message data from the original message is not included with the report message.

Do not specify more than one of MQRO_EXPIRATION, MQRO_EXPIRATION_WITH_DATA, and MQRO_EXPIRATION_WITH_FULL_DATA.

MQRO_EXPIRATION_WITH_DATA

Expiration reports with data required.

This is the same as MQRO_EXPIRATION, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of MQRO_EXPIRATION, MQRO_EXPIRATION_WITH_DATA, and MQRO_EXPIRATION_WITH_FULL_DATA.

MQRO_EXPIRATION_WITH_FULL_DATA

Expiration reports with full data required.

This is the same as MQRO_EXPIRATION, except that all of the application message data from the original message is included in the report message.

Do not specify more than one of MQRO_EXPIRATION, MQRO_EXPIRATION_WITH_DATA, and MQRO_EXPIRATION_WITH_FULL_DATA.

Confirm-on-arrival options: You can specify one of the options listed below to request a confirm-on-arrival report message.

MQRO_COA

Confirm-on-arrival reports required.

This type of report is generated by the queue manager that owns the destination queue, when the message is placed on the destination queue. Message data from the original message is not included with the report message.

If the message is put as part of a unit of work, and the destination queue is a local queue, the COA report message generated by the queue manager becomes available for retrieval only if and when the unit of work is committed.

A COA report is not generated if the *Format* field in the message descriptor is MQFMT_XMIT_Q_HEADER or MQFMT_DEAD_LETTER_HEADER. This prevents a COA report being generated if the message is put on a transmission queue, or is undeliverable and put on a dead-letter queue.

Do not specify more than one of MQRO_COA, MQRO_COA_WITH_DATA, and MQRO_COA_WITH_FULL_DATA.

MQRO_COA_WITH_DATA

Confirm-on-arrival reports with data required.

This is the same as MQRO_COA, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of MQRO_COA, MQRO_COA_WITH_DATA, and MQRO_COA_WITH_FULL_DATA.

MQRO_COA_WITH_FULL_DATA

Confirm-on-arrival reports with full data required.

This is the same as MQRO_COA, except that all of the application message data from the original message is included in the report message.

Do not specify more than one of MQRO_COA, MQRO_COA_WITH_DATA, and MQRO_COA_WITH_FULL_DATA.

Confirm-on-delivery options: You can specify one of the options listed below to request a confirm-on-delivery report message.

MQRO_COD

Confirm-on-delivery reports required.

This type of report is generated by the queue manager when an application retrieves the message from the destination queue in a way that causes the message to be deleted from the queue. Message data from the original message is not included with the report message.

If the message is retrieved as part of a unit of work, the report message is generated within the same unit of work, so that the report is not available until the unit of work is committed. If the unit of work is backed out, the report is not sent.

A COD report is not always generated if a message is retrieved with the MQGMO_MARK_SKIP_BACKOUT option. If the primary unit of work is backed out but the secondary unit of work is committed, the message is removed from the queue, but a COD report is not generated.

A COD report is not generated if the *Format* field in the message descriptor is MQFMT_DEAD_LETTER_HEADER. This prevents a COD report being generated if the message is undeliverable and put on a dead-letter queue.

MQRO_COD is not valid if the destination queue is an XCF queue.

Do not specify more than one of MQRO_COD, MQRO_COD_WITH_DATA, and MQRO_COD_WITH_FULL_DATA.

MQRO_COD_WITH_DATA

Confirm-on-delivery reports with data required.

This is the same as MQRO_COD, except that the first 100 bytes of the application message data from the original message are included in the

MQMD – Report field

report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

If MQGMO_ACCEPT_TRUNCATED_MSG is specified on the MQGET call for the original message, and the message retrieved is truncated, the amount of application message data placed in the report message depends on the environment:

- On z/OS, it is the minimum of:
 - The length of the original message
 - The length of the buffer used to retrieve the message
 - 100 bytes.
- In other environments, it is the minimum of:
 - The length of the original message
 - 100 bytes.

MQRO_COD_WITH_DATA is not valid if the destination queue is an XCF queue.

Do not specify more than one of MQRO_COD, MQRO_COD_WITH_DATA, and MQRO_COD_WITH_FULL_DATA.

MQRO_COD_WITH_FULL_DATA

Confirm-on-delivery reports with full data required.

This is the same as MQRO_COD, except that all of the application message data from the original message is included in the report message.

MQRO_COD_WITH_FULL_DATA is not valid if the destination queue is an XCF queue.

Do not specify more than one of MQRO_COD, MQRO_COD_WITH_DATA, and MQRO_COD_WITH_FULL_DATA.

Action-notification options: You can specify one or both of the options listed below to request that the receiving application send a positive-action or negative-action report message.

- On VSE/ESA, these options are not supported.

MQRO_PAN

Positive action notification reports required.

This type of report is generated by the application that retrieves the message and acts upon it. It indicates that the action requested in the message has been performed successfully. The application generating the report determines whether or not any data is to be included with the report.

Other than conveying this request to the application retrieving the message, the queue manager takes no action based upon this option. It is the responsibility of the retrieving application to generate the report if appropriate.

MQRO_NAN

Negative action notification reports required.

This type of report is generated by the application that retrieves the message and acts upon it. It indicates that the action requested in the message has *not* been performed successfully. The application generating the report determines whether or not any data is to be included with the

report. For example, it may be desirable to include some data indicating why the request could not be performed.

Other than conveying this request to the application retrieving the message, the queue manager takes no action based upon this option. It is the responsibility of the retrieving application to generate the report if appropriate.

Determination of which conditions correspond to a positive action and which correspond to a negative action is the responsibility of the application. However, it is recommended that if the request has been only partially performed, a NAN report rather than a PAN report should be generated if requested. It is also recommended that every possible condition should correspond to either a positive action, or a negative action, but not both.

Message-identifier options: You can specify one of the options listed below to control how the *MsgId* of the report message (or of the reply message) is to be set.

- On VSE/ESA, these options are not supported.

MQRO_NEW_MSG_ID

New message identifier.

This is the default action, and indicates that if a report or reply is generated as a result of this message, a new *MsgId* is to be generated for the report or reply message.

MQRO_PASS_MSG_ID

Pass message identifier.

If a report or reply is generated as a result of this message, the *MsgId* of this message is to be copied to the *MsgId* of the report or reply message.

If this option is not specified, MQRO_NEW_MSG_ID is assumed.

Correlation-identifier options: You can specify one of the options listed below to control how the *CorrelId* of the report message (or of the reply message) is to be set.

- On VSE/ESA, these options are not supported.

MQRO_COPY_MSG_ID_TO_CORREL_ID

Copy message identifier to correlation identifier.

This is the default action, and indicates that if a report or reply is generated as a result of this message, the *MsgId* of this message is to be copied to the *CorrelId* of the report or reply message.

MQRO_PASS_CORREL_ID

Pass correlation identifier.

If a report or reply is generated as a result of this message, the *CorrelId* of this message is to be copied to the *CorrelId* of the report or reply message.

If this option is not specified, MQRO_COPY_MSG_ID_TO_CORREL_ID is assumed.

Servers replying to requests or generating report messages are recommended to check whether the MQRO_PASS_MSG_ID or MQRO_PASS_CORREL_ID options were set in the original message. If they were, the servers should take the action described for those options. If neither is set, the servers should take the corresponding default action.

MQMD – Report field

Disposition options: You can specify one of the options listed below to control the disposition of the original message when it cannot be delivered to the destination queue. These options apply only to those situations that would result in an exception report message being generated if one had been requested by the sending application. The application can set the disposition options independently of requesting exception reports.

- On VSE/ESA, these options are not supported.

MQRO_DEAD_LETTER_Q

Place message on dead-letter queue.

This is the default action, and indicates that the message should be placed on the dead-letter queue, if the message cannot be delivered to the destination queue. An exception report message will be generated, if one was requested by the sender.

MQRO_DISCARD_MSG

Discard message.

This indicates that the message should be discarded if it cannot be delivered to the destination queue. An exception report message will be generated, if one was requested by the sender.

If it is desired to return the original message to the sender, without the original message being placed on the dead-letter queue, the sender should specify MQRO_DISCARD_MSG with MQRO_EXCEPTION_WITH_FULL_DATA.

Default option: You can specify the following if no report options are required:

MQRO_NONE

No reports required.

This value can be used to indicate that no other options have been specified. MQRO_NONE is defined to aid program documentation. It is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

General information:

1. All report types required must be specifically requested by the application sending the original message. For example, if a COA report is requested but an exception report is not, a COA report is generated when the message is placed on the destination queue, but no exception report is generated if the destination queue is full when the message arrives there. If no *Report* options are set, no report messages are generated by the queue manager or message channel agent (MCA).

Some report options can be specified even though the local queue manager does not recognize them; this is useful when the option is to be processed by the *destination* queue manager. See Appendix E, “Report options and message flags”, on page 575 for more details.

If a report message is requested, the name of the queue to which the report should be sent must be specified in the *ReplyToQ* field. When a report message is received, the nature of the report can be determined by examining the *Feedback* field in the message descriptor.

2. If the queue manager or MCA that generates a report message is unable to put the report message on the reply queue (for example, because the reply queue or transmission queue is full), the report message is placed instead on the

dead-letter queue. If that *also* fails, or there is no dead-letter queue, the action taken depends on the type of the report message:

- If the report message is an exception report, the message which caused the exception report to be generated is left on its transmission queue; this ensures that the message is not lost.
- For all other report types, the report message is discarded and processing continues normally. This is done because either the original message has already been delivered safely (for COA or COD report messages), or is no longer of any interest (for an expiration report message).

Once a report message has been placed successfully on a queue (either the destination queue or an intermediate transmission queue), the message is no longer subject to special processing — it is treated just like any other message.

3. When the report is generated, the *ReplyToQ* queue is opened and the report message put using the authority of the *UserIdentifier* in the MQMD of the message causing the report, except in the following cases:
 - Exception reports generated by a receiving MCA are put with whatever authority the MCA used when it tried to put the message causing the report. The *PutAuthority* channel attribute determines the user identifier used.
 - COA reports generated by the queue manager are put with whatever authority was used when the message causing the report was put on the queue manager generating the report. For example, if the message was put by a receiving MCA using the MCA's user identifier, the queue manager puts the COA report using the MCA's user identifier.

Applications generating reports should normally use the same authority as they would have used to generate a reply; this should normally be the authority of the user identifier in the original message.

If the report has to travel to a remote destination, senders and receivers can decide whether or not to accept it, in the same way as they do for other messages.

4. If a report message with data is requested:
 - The report message is always generated with the amount of data requested by the sender of the original message. If the report message is too big for the reply queue, the processing described above occurs; the report message is never truncated in order to fit on the reply queue.
 - If the *Format* of the original message is *MQFMT_XMIT_Q_HEADER*, the data included in the report does not include the *MQXQH*. The report data starts with the first byte of the data beyond the *MQXQH* in the original message. This occurs whether or not the queue is a transmission queue.
5. If a COA, COD, or expiration report message is received at the reply queue, it is guaranteed that the original message arrived, was delivered, or expired, as appropriate. However, if one or more of these report messages is requested and is *not* received, the reverse cannot be assumed, since one of the following may have occurred:
 - a. The report message is held up because a link is down.
 - b. The report message is held up because a blocking condition exists at an intermediate transmission queue or at the reply queue (for example, the queue is full or inhibited for puts).
 - c. The report message is on a dead-letter queue.
 - d. When the queue manager was attempting to generate the report message, it was unable to put it on the appropriate queue, and was also unable to put it on the dead-letter queue, so the report message could not be generated.

MQMD – Report field

- e. A failure of the queue manager occurred between the action being reported (arrival, delivery or expiry), and generation of the corresponding report message. (This does not happen for COD report messages if the application retrieves the original message within a unit of work, as the COD report message is generated within the same unit of work.)

Exception report messages may be held up in the same way for reasons 1, 2, and 3 above. However, when an MCA is unable to generate an exception report message (the report message cannot be put either on the reply queue or the dead-letter queue), the original message remains on the transmission queue at the sender, and the channel is closed. This occurs irrespective of whether the report message was to be generated at the sending or the receiving end of the channel.

6. If the original message is temporarily blocked (resulting in an exception report message being generated and the original message being put on a dead-letter queue), but the blockage clears and an application then reads the original message from the dead-letter queue and puts it again to its destination, the following may occur:
 - Even though an exception report message has been generated, the original message eventually arrives successfully at its destination.
 - More than one exception report message is generated in respect of a single original message, since the original message may encounter another blockage later.

Report messages for message segments:

1. Report messages can be requested for messages that have segmentation allowed (see the description of the MQMF_SEGMENTATION_ALLOWED flag). If the queue manager finds it necessary to segment the message, a report message can be generated for each of the segments that subsequently encounters the relevant condition. Applications should therefore be prepared to receive multiple report messages for each type of report message requested. The *GroupId* field in the report message can be used to correlate the multiple reports with the group identifier of the original message, and the *Feedback* field used to identify the type of each report message.
2. If MQGMO_LOGICAL_ORDER is used to retrieve report messages for segments, be aware that reports of *different types* may be returned by the successive MQGET calls. For example, if both COA and COD reports are requested for a message that is segmented by the queue manager, the MQGET calls for the report messages may return the COA and COD report messages interleaved in an unpredictable fashion. This can be avoided by using the MQGMO_COMPLETE_MSG option (optionally with MQGMO_ACCEPT_TRUNCATED_MSG). MQGMO_COMPLETE_MSG causes the queue manager to reassemble report messages that have the same report type. For example, the first MQGET call might reassemble all of the COA messages relating to the original message, and the second MQGET call might reassemble all of the COD messages. Which is reassembled first depends on which type of report message happens to occur first on the queue.
3. Applications that themselves put segments can specify different report options for each segment. However, the following points should be noted:
 - If the segments are retrieved using the MQGMO_COMPLETE_MSG option, only the report options in the *first* segment are honored by the queue manager.
 - If the segments are retrieved one by one, and most of them have one of the MQRO_COD_* options, but at least one segment does not, it will not be

possible to use the MQGMO_COMPLETE_MSG option to retrieve the report messages with a single MQGET call, or use the MQGMO_ALL_SEGMENTS_AVAILABLE option to detect when all of the report messages have arrived.

- In an MQ network, it is possible for the queue managers to have differing capabilities. If a report message for a segment is generated by a queue manager or MCA that does not support segmentation, the queue manager or MCA will not by default include the necessary segment information in the report message, and this may make it difficult to identify the original message that caused the report to be generated. This difficulty can be avoided by requesting data with the report message, that is, by specifying the appropriate MQRO_*_WITH_DATA or MQRO_*_WITH_FULL_DATA options. However, be aware that if MQRO_*_WITH_DATA is specified, *less than* 100 bytes of application message data may be returned to the application which retrieves the report message, if the report message is generated by a queue manager or MCA that does not support segmentation.

Contents of the message descriptor for a report message: When the queue manager or message channel agent (MCA) generates a report message, it sets the fields in the message descriptor to the following values, and then puts the message in the normal way.

Field in MQMD	Value used
<i>StrucId</i>	MQMD_STRUC_ID
<i>Version</i>	MQMD_VERSION_2
<i>Report</i>	MQRO_NONE
<i>MsgType</i>	MQMT_REPORT
<i>Expiry</i>	MQEI_UNLIMITED
<i>Feedback</i>	As appropriate for the nature of the report (MQFB_COA, MQFB_COD, MQFB_EXPIRATION, or an MQRC_* value)
<i>Encoding</i>	Copied from the original message descriptor
<i>CodedCharSetId</i>	Copied from the original message descriptor
<i>Format</i>	Copied from the original message descriptor
<i>Priority</i>	Copied from the original message descriptor
<i>Persistence</i>	Copied from the original message descriptor
<i>MsgId</i>	As specified by the report options in the original message descriptor
<i>CorrelId</i>	As specified by the report options in the original message descriptor
<i>BackoutCount</i>	0
<i>ReplyToQ</i>	Blanks
<i>ReplyToQMGr</i>	Name of queue manager
<i>UserIdentifier</i>	As set by the MQPMO_PASS_IDENTITY_CONTEXT option
<i>AccountingToken</i>	As set by the MQPMO_PASS_IDENTITY_CONTEXT option
<i>ApplIdentityData</i>	As set by the MQPMO_PASS_IDENTITY_CONTEXT option
<i>PutApplType</i>	MQAT_QMGR, or as appropriate for the message channel agent
<i>PutApplName</i>	First 28 bytes of the queue-manager name or message channel agent name. For report messages generated by the IMS bridge, this field contains the XCF group name and XCF member name of the IMS system to which the message relates.
<i>PutDate</i>	Date when report message is sent
<i>PutTime</i>	Time when report message is sent
<i>ApplOriginData</i>	Blanks
<i>GroupId</i>	Copied from the original message descriptor

MQMD – Report field

Field in MQMD	Value used
<i>MsgSeqNumber</i>	Copied from the original message descriptor
<i>Offset</i>	Copied from the original message descriptor
<i>MsgFlags</i>	Copied from the original message descriptor
<i>OriginalLength</i>	Copied from the original message descriptor if not MQOL_UNDEFINED, and set to the length of the original message data otherwise

An application generating a report is recommended to set similar values, except for the following:

- The *ReplyToQMGr* field can be set to blanks (the queue manager will change this to the name of the local queue manager when the message is put).
- The context fields should be set using the option that would have been used for a reply, normally MQPMO_PASS_IDENTITY_CONTEXT.

Analyzing the report field: The *Report* field contains subfields; because of this, applications that need to check whether the sender of the message requested a particular report should use one of the techniques described in “Analyzing the report field” on page 576.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQRO_NONE.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQMD_STRUC_ID

Identifier for message descriptor structure.

For the C programming language, the constant MQMD_STRUC_ID_ARRAY is also defined; this has the same value as MQMD_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQMD_STRUC_ID.

UserIdentifier (MQCHAR12)

User identifier.

This is part of the **identity context** of the message. For more information about message context, see “Overview” on page 141; also see the *WebSphere MQ Application Programming Guide*.

UserIdentifier specifies the user identifier of the application that originated the message. The queue manager treats this information as character data, but does not define the format of it.

After a message has been received, *UserIdentifier* can be used in the *AlternateUserId* field of the *ObjDesc* parameter of a subsequent MQOPEN or MQPUT1 call, so that the authorization check is performed for the *UserIdentifier* user instead of the application performing the open.

When the queue manager generates this information for an MQPUT or MQPUT1 call:

- On z/OS, the queue manager uses the *AlternateUserId* from the *ObjDesc* parameter of the MQOPEN or MQPUT1 call if the MQOO_ALTERNATE_USER_AUTHORITY or MQPMO_ALTERNATE_USER_AUTHORITY option was specified. If the relevant option was not specified, the queue manager uses a user identifier determined from the environment.
- In other environments, the queue manager always uses a user identifier determined from the environment.

When the user identifier is determined from the environment:

- On z/OS, the queue manager uses:
 - For MVS (batch), the user identifier from the JES JOB card or started task
 - For TSO, the user identifier propagated to the job during job submission
 - For CICS, the user identifier associated with the task
 - For IMS, the user identifier depends on the type of application:
 - For:
 - Nonmessage BMP regions
 - Nonmessage IFP regions
 - Message BMP and message IFP regions that have *not* issued a successful GU call

the queue manager uses the user identifier from the region JES JOB card or the TSO user identifier. If these are blank or null, it uses the name of the program specification block (PSB).
 - For:
 - Message BMP and message IFP regions that *have* issued a successful GU call
 - MPP regions

the queue manager uses one of:

 - The signed-on user identifier associated with the message
 - The logical terminal (LTERM) name
 - The user identifier from the region JES JOB card
 - The TSO user identifier
 - The PSB name
- On OS/2, the queue manager uses the string “os2”.
- On OS/400, the queue manager uses the name of the user profile associated with the application job.
- On Compaq NonStop Kernel, the value of the user identifier, when set by the queue manager during an MQPUT or MQPUT1, is the MQSeries Principal name found in the queue manager’s Principal database corresponding to the effective user identifier of the application.
- On Compaq OpenVMS Alpha and UNIX systems, the queue manager uses:
 - The application’s logon name
 - The effective user identifier of the process if no logon is available
 - The user identifier associated with the transaction, if the application is a CICS transaction
- On VSE/ESA, this is a reserved field.
- On Windows 3.1, the queue manager uses the string “WINDOWS”.
- On Windows 95, Windows 98, and Windows, the queue manager uses the first 12 characters of the logged-on user name.

MQMD – UserIdentifier field

This field is normally an output field generated by the queue manager but for an MQPUT or MQPUT1 call you can make this field to be an input/output field and specify the UserIdentification field instead of letting the queue manager generate this information. You should specify either MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT in the PutMsgOpts parameter and specify a userid in the UserIdentifier field if you do not want the queue manager to generate the UserIdentifier field for an MQPUT or MQPUT1 call.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. Any information following a null character within the field is discarded. The null character and any following characters are converted to blanks by the queue manager. If MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *UserIdentifier* that was transmitted with the message. If the message has no context, the field is entirely blank.

This is an output field for the MQGET call. The length of this field is given by MQ_USER_ID_LENGTH. The initial value of this field is the null string in C, and 12 blank characters in other programming languages.

Version (MQLONG)

Structure version number.

The value must be one of the following:

MQMD_VERSION_1

Version-1 message descriptor structure.

This version is supported in all environments.

MQMD_VERSION_2

Version-2 message descriptor structure.

This version is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Note: When a version-2 MQMD is used, the queue manager performs additional checks on any MQ header structures that may be present at the beginning of the application message data; for further details see usage note 4 on page 433 for the MQPUT call.

Fields that exist only in the more-recent version of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQMD_CURRENT_VERSION

Current version of message descriptor structure.

This is always an input field. The initial value of this field is MQMD_VERSION_1.

Initial values and language declarations

Table 52. Initial values of fields in MQMD

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQMD_STRUC_ID	'MDbb '
<i>Version</i>	MQMD_VERSION_1	1
<i>Report</i>	MQRO_NONE	0
<i>MsgType</i>	MQMT_DATAGRAM	8
<i>Expiry</i>	MQEI_UNLIMITED	-1
<i>Feedback</i>	MQFB_NONE	0
<i>Encoding</i>	MQENC_NATIVE	Depends on environment
<i>CodedCharSetId</i>	MQCCSI_Q_MGR	0
<i>Format</i>	MQFMT_NONE	Blanks
<i>Priority</i>	MQPRI_PRIORITY_AS_Q_DEF	-1
<i>Persistence</i>	MQPER_PERSISTENCE_AS_Q_DEF	2
<i>MsgId</i>	MQMI_NONE	Nulls
<i>CorrelId</i>	MQCI_NONE	Nulls
<i>BackoutCount</i>	None	0
<i>ReplyToQ</i>	None	Null string or blanks
<i>ReplyToQMgr</i>	None	Null string or blanks
<i>UserIdentifier</i>	None	Null string or blanks
<i>AccountingToken</i>	MQACT_NONE	Nulls
<i>ApplIdentityData</i>	None	Null string or blanks
<i>PutApplType</i>	MQAT_NO_CONTEXT	0
<i>PutApplName</i>	None	Null string or blanks
<i>PutDate</i>	None	Null string or blanks
<i>PutTime</i>	None	Null string or blanks
<i>ApplOriginData</i>	None	Null string or blanks
<i>GroupId</i>	MQGI_NONE	Nulls
<i>MsgSeqNumber</i>	None	1
<i>Offset</i>	None	0
<i>MsgFlags</i>	MQMF_NONE	0
<i>OriginalLength</i>	MQOL_UNDEFINED	-1

Notes:

1. The symbol 'b' represents a single blank character.
2. The value 'Null string or blanks' denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQMD_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQMD MyMD = {MQMD_DEFAULT};
```

MQMD – Language declarations

C declaration

```
typedef struct tagMQMD MQMD;
struct tagMQMD {
    MQCHAR4  StrucId;           /* Structure identifier */
    MQLONG   Version;         /* Structure version number */
    MQLONG   Report;         /* Options for report messages */
    MQLONG   MsgType;        /* Message type */
    MQLONG   Expiry;         /* Message lifetime */
    MQLONG   Feedback;       /* Feedback or reason code */
    MQLONG   Encoding;       /* Numeric encoding of message data */
    MQLONG   CodedCharSetId; /* Character set identifier of message
                             data */
    MQCHAR8  Format;          /* Format name of message data */
    MQLONG   Priority;        /* Message priority */
    MQLONG   Persistence;    /* Message persistence */
    MQBYTE24 MsgId;          /* Message identifier */
    MQBYTE24 CorrelId;       /* Correlation identifier */
    MQLONG   BackoutCount;   /* Backout counter */
    MQCHAR48 ReplyToQ;       /* Name of reply queue */
    MQCHAR48 ReplyToQMGr;    /* Name of reply queue manager */
    MQCHAR12 UserIdentifier; /* User identifier */
    MQBYTE32 AccountingToken; /* Accounting token */
    MQCHAR32 ApplIdentityData; /* Application data relating to
                             identity */
    MQLONG   PutAppIType;    /* Type of application that put the
                             message */
    MQCHAR28 PutAppIName;    /* Name of application that put the
                             message */
    MQCHAR8  PutDate;        /* Date when message was put */
    MQCHAR8  PutTime;        /* Time when message was put */
    MQCHAR4  ApplOriginData; /* Application data relating to origin */
    MQBYTE24 GroupId;        /* Group identifier */
    MQLONG   MsgSeqNumber;   /* Sequence number of logical message
                             within group */
    MQLONG   Offset;         /* Offset of data in physical message
                             from start of logical message */
    MQLONG   MsgFlags;       /* Message flags */
    MQLONG   OriginalLength; /* Length of original message */
};
```

COBOL declaration

```
** MQMD structure
10 MQMD.
** Structure identifier
15 MQMD-STRUCID PIC X(4).
** Structure version number
15 MQMD-VERSION PIC S9(9) BINARY.
** Options for report messages
15 MQMD-REPORT PIC S9(9) BINARY.
** Message type
15 MQMD-MSGTYPE PIC S9(9) BINARY.
** Message lifetime
15 MQMD-EXPIRY PIC S9(9) BINARY.
** Feedback or reason code
15 MQMD-FEEDBACK PIC S9(9) BINARY.
** Numeric encoding of message data
15 MQMD-ENCODING PIC S9(9) BINARY.
** Character set identifier of message data
15 MQMD-CODEDCHARSETID PIC S9(9) BINARY.
** Format name of message data
15 MQMD-FORMAT PIC X(8).
** Message priority
15 MQMD-PRIORITY PIC S9(9) BINARY.
** Message persistence
15 MQMD-PERSISTENCE PIC S9(9) BINARY.
```

MQMD – Language declarations

```
** Message identifier
15 MQMD-MSGID PIC X(24).
** Correlation identifier
15 MQMD-CORRELID PIC X(24).
** Backout counter
15 MQMD-BACKOUTCOUNT PIC S9(9) BINARY.
** Name of reply queue
15 MQMD-REPLYTOQ PIC X(48).
** Name of reply queue manager
15 MQMD-REPLYTOQMGR PIC X(48).
** User identifier
15 MQMD-USERIDENTIFIER PIC X(12).
** Accounting token
15 MQMD-ACCOUNTINGTOKEN PIC X(32).
** Application data relating to identity
15 MQMD-APPLIDENTITYDATA PIC X(32).
** Type of application that put the message
15 MQMD-PUTAPPLTYPE PIC S9(9) BINARY.
** Name of application that put the message
15 MQMD-PUTAPPLNAME PIC X(28).
** Date when message was put
15 MQMD-PUTDATE PIC X(8).
** Time when message was put
15 MQMD-PUTTIME PIC X(8).
** Application data relating to origin
15 MQMD-APPLORIGINDATA PIC X(4).
** Group identifier
15 MQMD-GROUPID PIC X(24).
** Sequence number of logical message within group
15 MQMD-MSGSEQNUMBER PIC S9(9) BINARY.
** Offset of data in physical message from start of logical message
15 MQMD-OFFSET PIC S9(9) BINARY.
** Message flags
15 MQMD-MSGFLAGS PIC S9(9) BINARY.
** Length of original message
15 MQMD-ORIGINALLENGTH PIC S9(9) BINARY.
```

PL/I declaration

```
dc1
1 MQMD based,
3 StrucId char(4), /* Structure identifier */
3 Version fixed bin(31), /* Structure version number */
3 Report fixed bin(31), /* Options for report messages */
3 MsgType fixed bin(31), /* Message type */
3 Expiry fixed bin(31), /* Message lifetime */
3 Feedback fixed bin(31), /* Feedback or reason code */
3 Encoding fixed bin(31), /* Numeric encoding of message
data */
3 CodedCharSetId fixed bin(31), /* Character set identifier of
message data */
3 Format char(8), /* Format name of message data */
3 Priority fixed bin(31), /* Message priority */
3 Persistence fixed bin(31), /* Message persistence */
3 MsgId char(24), /* Message identifier */
3 CorrelId char(24), /* Correlation identifier */
3 BackoutCount fixed bin(31), /* Backout counter */
3 ReplyToQ char(48), /* Name of reply queue */
3 ReplyToQMgr char(48), /* Name of reply queue manager */
3 UserIdentifier char(12), /* User identifier */
3 AccountingToken char(32), /* Accounting token */
3 ApplIdentityData char(32), /* Application data relating to
identity */
3 PutAppIType fixed bin(31), /* Type of application that put the
message */
3 PutAppIName char(28), /* Name of application that put the
message */
```

MQMD – Language declarations

```
3 PutDate          char(8),      /* Date when message was put */
3 PutTime          char(8),      /* Time when message was put */
3 ApplOriginData   char(4),      /* Application data relating to
                                origin */
3 GroupId          char(24),     /* Group identifier */
3 MsgSeqNumber     fixed bin(31), /* Sequence number of logical
                                message within group */
3 Offset           fixed bin(31), /* Offset of data in physical
                                message from start of logical
                                message */
3 MsgFlags         fixed bin(31), /* Message flags */
3 OriginalLength   fixed bin(31); /* Length of original message */
```

System/390 assembler declaration

```
MQMD                DSECT
MQMD_STRUCID        DS    CL4    Structure identifier
MQMD_VERSION        DS    F      Structure version number
MQMD_REPORT         DS    F      Options for report messages
MQMD_MSGTYPE        DS    F      Message type
MQMD_EXPIRY         DS    F      Message lifetime
MQMD_FEEDBACK       DS    F      Feedback or reason code
MQMD_ENCODING       DS    F      Numeric encoding of message data
MQMD_CODEDCHARSETID DS    F      Character set identifier of message
*
MQMD_FORMAT         DS    CL8    Format name of message data
MQMD_PRIORITY       DS    F      Message priority
MQMD_PERSISTENCE    DS    F      Message persistence
MQMD_MSGID          DS    XL24   Message identifier
MQMD_CORRELID       DS    XL24   Correlation identifier
MQMD_BACKOUTCOUNT DS    F      Backout counter
MQMD_REPLYTOQ       DS    CL48   Name of reply queue
MQMD_REPLYTOQMGR    DS    CL48   Name of reply queue manager
MQMD_USERIDENTIFIER DS    CL12   User identifier
MQMD_ACCOUNTINGTOKEN DS    XL32  Accounting token
MQMD_APPLIDENTITYDATA DS    CL32  Application data relating to identity
MQMD_PUTAPPLTYPE    DS    F      Type of application that put the
*
MQMD_PUTAPPLNAME    DS    CL28   Name of application that put the
*
MQMD_PUTDATE        DS    CL8    Date when message was put
MQMD_PUTTIME        DS    CL8    Time when message was put
MQMD_APPLORIGINDATA DS    CL4    Application data relating to origin
MQMD_GROUPID        DS    XL24   Group identifier
MQMD_MSGSEQUENBER   DS    F      Sequence number of logical message
*
MQMD_OFFSET         DS    F      Offset of data in physical message
*
MQMD_MSGFLAGS       DS    F      Message flags
MQMD_ORIGINALLENGTH DS    F      Length of original message
*
MQMD_LENGTH         EQU    *-MQMD
                    ORG    MQMD
MQMD_AREA           DS    CL(MQMD_LENGTH)
```

TAL declaration

```
STRUCT      MQMD^DEF (*);
  BEGIN
  STRUCT      STRUCID;
  BEGIN STRING BYTE [0:3]; END;
  INT(32)     VERSION;
  INT(32)     REPORTOPTIONS;
  INT(32)     MSGTYPE;
  INT(32)     EXPIRY;
  INT(32)     FEEDBACK;
  INT(32)     ENCODING;
```

```

INT(32)          CODEDCHARSETID;
STRUCT          FORMAT;
BEGIN STRING BYTE [0:7]; END;
INT(32)          PRIORITY;
INT(32)          PERSISTENCE;
STRUCT          MSGID;
BEGIN STRING BYTE [0:23]; END;
STRUCT          CORRELID;
BEGIN STRING BYTE [0:23]; END;
INT(32)          BACKOUTCOUNT;
STRUCT          REPLYTOQ;
BEGIN STRING BYTE [0:47]; END;
STRUCT          REPLYTOQMGR;
BEGIN STRING BYTE [0:47]; END;
STRUCT          USERIDENTIFIER;
BEGIN STRING BYTE [0:11]; END;
STRUCT          ACCOUNTINGTOKEN;
BEGIN STRING BYTE [0:31]; END;
STRUCT          APPLIDENTITYDATA;
BEGIN STRING BYTE [0:31]; END;
INT(32)          PUTAPPLTYPE;
STRUCT          PUTAPPLNAME;
BEGIN STRING BYTE [0:27]; END;
STRUCT          PUTDATE;
BEGIN STRING BYTE [0:7]; END;
STRUCT          PUTTIME;
BEGIN STRING BYTE [0:7]; END;
STRUCT          APPLORIGINDATA;
BEGIN STRING BYTE [0:3]; END;
END;

```

Visual Basic declaration

```

Type MQMD
  StrucId      As String*4  'Structure identifier'
  Version     As Long      'Structure version number'
  Report      As Long      'Options for report messages'
  MsgType     As Long      'Message type'
  Expiry      As Long      'Message lifetime'
  Feedback    As Long      'Feedback or reason code'
  Encoding    As Long      'Numeric encoding of message data'
  CodedCharSetId As Long    'Character set identifier of message'
  data

  Format      As String*8   'Format name of message data'
  Priority    As Long      'Message priority'
  Persistence As Long      'Message persistence'
  MsgId      As MQBYTE24   'Message identifier'
  CorrelId   As MQBYTE24   'Correlation identifier'
  BackoutCount As Long     'Backout counter'
  ReplyToQ   As String*48  'Name of reply queue'
  ReplyToQMgr As String*48 'Name of reply queue manager'
  UserIdentifier As String*12 'User identifier'
  AccountingToken As MQBYTE32 'Accounting token'
  ApplIdentityData As String*32 'Application data relating to identity'
  PutApplType As Long      'Type of application that put the'
  message

  PutApplName As String*28 'Name of application that put the'
  message

  PutDate    As String*8   'Date when message was put'
  PutTime    As String*8   'Time when message was put'
  ApplOriginData As String*4 'Application data relating to origin'
  GroupId    As MQBYTE24   'Group identifier'
  MsgSeqNumber As Long     'Sequence number of logical message'
  within group

  Offset     As Long      'Offset of data in physical message'

```

MQMD – Language declarations

MsgFlags	As Long	'from start of logical message'
OriginalLength	As Long	'Message flags'
End Type		'Length of original message'

Chapter 11. MQMDE – Message descriptor extension

The following table summarizes the fields in the structure.

Table 53. Fields in MQMDE

Field	Description	Page
<i>StrucId</i>	Structure identifier	207
<i>Version</i>	Structure version number	208
<i>StrucLength</i>	Length of MQMDE structure	207
<i>Encoding</i>	Numeric encoding of data that follows MQMDE	206
<i>CodedCharSetId</i>	Character set identifier of data that follows MQMDE	205
<i>Format</i>	Format name of data that follows MQMDE	206
<i>Flags</i>	General flags	206
<i>GroupId</i>	Group identifier	207
<i>MsgSeqNumber</i>	Sequence number of logical message within group	207
<i>Offset</i>	Offset of data in physical message from start of logical message	207
<i>MsgFlags</i>	Message flags	207
<i>OriginalLength</i>	Length of original message	207

Overview

Availability: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Purpose: The MQMDE structure describes the data that sometimes occurs preceding the application message data. The structure contains those MQMD fields that exist in the version-2 MQMD, but not in the version-1 MQMD.

Format name: MQFMT_MD_EXTENSION.

Character set and encoding: Data in MQMDE must be in the character set and encoding of the local queue manager; these are given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE for the C programming language, respectively.

The character set and encoding of the MQMDE must be set into the *CodedCharSetId* and *Encoding* fields in:

- The MQMD (if the MQMDE structure is at the start of the message data), or
- The header structure that precedes the MQMDE structure (all other cases).

If the MQMDE is not in the queue manager's character set and encoding, the MQMDE is accepted but not honored, that is, the MQMDE is treated as message data.

MQMDE – Message descriptor extension

Note: On OS/2 and Windows, applications compiled with Micro Focus COBOL use a value of MQENC_NATIVE that is different from the queue-manager's encoding. Although numeric fields in the MQMD structure on the MQPUT, MQPUT1, and MQGET calls must be in the Micro Focus COBOL encoding, numeric fields in the MQMDE structure must be in the queue-manager's encoding. This latter is given by MQENC_NATIVE for the C programming language, and has the value 546.

Usage: Normal applications should use a version-2 MQMD, in which case they will not encounter an MQMDE structure. However, specialized applications, and applications that continue to use a version-1 MQMD, may encounter an MQMDE in some situations. The MQMDE structure can occur in the following circumstances:

- Specified on the MQPUT and MQPUT1 calls
- Returned by the MQGET call
- In messages on transmission queues

These are described below.

MQMDE specified on MQPUT and MQPUT1 calls: On the MQPUT and MQPUT1 calls, if the application provides a version-1 MQMD, the application can optionally prefix the message data with an MQMDE, setting the *Format* field in MQMD to MQFMT_MD_EXTENSION to indicate that an MQMDE is present. If the application does not provide an MQMDE, the queue manager assumes default values for the fields in the MQMDE. The default values that the queue manager uses are the same as the initial values for the structure – see Table 55 on page 208.

If the application provides a version-2 MQMD *and* prefixes the application message data with an MQMDE, the structures are processed as shown in Table 54.

Table 54. Queue-manager action when MQMDE specified on MQPUT or MQPUT1

MQMD version	Values of version-2 fields	Values of corresponding fields in MQMDE	Action taken by queue manager
1	–	Valid	MQMDE is honored
2	Default	Valid	MQMDE is honored
2	Not default	Valid	MQMDE is treated as message data
1 or 2	Any	Not valid	Call fails with an appropriate reason code
1 or 2	Any	MQMDE is in the wrong character set or encoding, or is an unsupported version	MQMDE is treated as message data

Note: On z/OS, if the application specifies a version-1 MQMD with an MQMDE, the queue manager validates the MQMDE only if the queue has an *IndexType* of MQIT_GROUP_ID.

There is one special case. If the application uses a version-2 MQMD to put a message that is a segment (that is, the MQMF_SEGMENT or MQMF_LAST_SEGMENT flag is set), and the format name in the MQMD is MQFMT_DEAD_LETTER_HEADER, the queue manager generates an MQMDE structure and inserts it *between* the MQDLH structure and the data that follows it. In the MQMD that the queue manager retains with the message, the version-2 fields are set to their default values.

Several of the fields that exist in the version-2 MQMD but not the version-1 MQMD are input/output fields on MQPUT and MQPUT1. However, the queue manager does *not* return any values in the equivalent fields in the MQMDE on

MQMDE – Message descriptor extension

output from the MQPUT and MQPUT1 calls; if the application requires those output values, it must use a version-2 MQMD.

MQMDE returned by MQGET call: On the MQGET call, if the application provides a version-1 MQMD, the queue manager prefixes the message returned with an MQMDE, but only if one or more of the fields in the MQMDE has a nondefault value. The queue manager sets the *Format* field in MQMD to the value MQFMT_MD_EXTENSION to indicate that an MQMDE is present.

If the application provides an MQMDE at the start of the *Buffer* parameter, the MQMDE is ignored. On return from the MQGET call, it is replaced by the MQMDE for the message (if one is needed), or overwritten by the application message data (if the MQMDE is not needed).

If an MQMDE is returned by the MQGET call, the data in the MQMDE is usually in the queue manager's character set and encoding. However the MQMDE may be in some other character set and encoding if:

- The MQMDE was treated as data on the MQPUT or MQPUT1 call (see Table 54 on page 204 for the circumstances that can cause this).
- The message was received from a remote queue manager connected by a TCP connection, and the receiving message channel agent (MCA) was not set up correctly (see the *WebSphere MQ Intercommunication* manual for further information).

Note: On OS/2 and Windows, applications compiled with Micro Focus COBOL use a value of MQENC_NATIVE that is different from the queue-manager's encoding (see above).

MQMDE in messages on transmission queues: Messages on transmission queues are prefixed with the MQXQH structure, which contains within it a version-1 MQMD. An MQMDE may also be present, positioned between the MQXQH structure and application message data, but it will usually be present only if one or more of the fields in the MQMDE has a nondefault value.

Other MQ header structures can also occur between the MQXQH structure and the application message data. For example, when the dead-letter header MQDLH is present, and the message is not a segment, the order is:

- MQXQH (containing a version-1 MQMD)
- MQMDE
- MQDLH
- application message data

Fields

The MQMDE structure contains the following fields; the fields are described in **alphabetic order**:

CodedCharSetId (MQLONG)

Character-set identifier of data that follows MQMDE.

This specifies the character set identifier of the data that follows the MQMDE structure; it does not apply to character data in the MQMDE structure itself.

MQMDE – CodedCharSetId field

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that this field is valid. The following special value can be used:

MQCCSI_INHERIT

Inherit character-set identifier of this structure.

Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

MQCCSI_INHERIT cannot be used if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

This value is supported in the following environments: AIX, HP-UX, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG)

Numeric encoding of data that follows MQMDE.

This specifies the numeric encoding of the data that follows the MQMDE structure; it does not apply to numeric data in the MQMDE structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that the field is valid. See the *Encoding* field described in Chapter 10, “MQMD – Message descriptor”, on page 141 for more information about data encodings.

The initial value of this field is MQENC_NATIVE.

Flags (MQLONG)

General flags.

The following flag can be specified:

MQMDEF_NONE

No flags.

The initial value of this field is MQMDEF_NONE.

Format (MQCHAR8)

Format name of data that follows MQMDE.

This specifies the format name of the data that follows the MQMDE structure.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that this field is valid. See the *Format* field described in Chapter 10, “MQMD – Message descriptor”, on page 141 for more information about format names.

The initial value of this field is MQFMT_NONE.

GroupId (MQBYTE24)

Group identifier.

See the *GroupId* field described in Chapter 10, “MQMD – Message descriptor”, on page 141. The initial value of this field is MQGI_NONE.

MsgFlags (MQLONG)

Message flags.

See the *MsgFlags* field described in Chapter 10, “MQMD – Message descriptor”, on page 141. The initial value of this field is MQMF_NONE.

MsgSeqNumber (MQLONG)

Sequence number of logical message within group.

See the *MsgSeqNumber* field described in Chapter 10, “MQMD – Message descriptor”, on page 141. The initial value of this field is 1.

Offset (MQLONG)

Offset of data in physical message from start of logical message.

See the *Offset* field described in Chapter 10, “MQMD – Message descriptor”, on page 141. The initial value of this field is 0.

OriginalLength (MQLONG)

Length of original message.

See the *OriginalLength* field described in Chapter 10, “MQMD – Message descriptor”, on page 141. The initial value of this field is MQOL_UNDEFINED.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQMDE_STRUC_ID

Identifier for message descriptor extension structure.

For the C programming language, the constant MQMDE_STRUC_ID_ARRAY is also defined; this has the same value as MQMDE_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQMDE_STRUC_ID.

StrucLength (MQLONG)

Length of MQMDE structure.

The following value is defined:

MQMDE_LENGTH_2

Length of version-2 message descriptor extension structure.

The initial value of this field is MQMDE_LENGTH_2.

MQMDE – Version field

Version (MQLONG)

Structure version number.

The value must be:

MQMDE_VERSION_2

Version-2 message descriptor extension structure.

The following constant specifies the version number of the current version:

MQMDE_CURRENT_VERSION

Current version of message descriptor extension structure.

The initial value of this field is MQMDE_VERSION_2.

Initial values and language declarations

Table 55. Initial values of fields in MQMDE

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQMDE_STRUC_ID	'MDEb'
<i>Version</i>	MQMDE_VERSION_2	2
<i>StrucLength</i>	MQMDE_LENGTH_2	72
<i>Encoding</i>	MQENC_NATIVE	Depends on environment
<i>CodedCharSetId</i>	MQCCSI_UNDEFINED	0
<i>Format</i>	MQFMT_NONE	Blanks
<i>Flags</i>	MQMDEF_NONE	0
<i>GroupId</i>	MQGI_NONE	Nulls
<i>MsgSeqNumber</i>	None	1
<i>Offset</i>	None	0
<i>MsgFlags</i>	MQMF_NONE	0
<i>OriginalLength</i>	MQOL_UNDEFINED	-1

Notes:

1. The symbol 'b' represents a single blank character.
2. In the C programming language, the macro variable MQMDE_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQMDE MyMDE = {MQMDE_DEFAULT};
```

C declaration

```
typedef struct tagMQMDE MQMDE;
struct tagMQMDE {
    MQCHAR4  StrucId;          /* Structure identifier */
    MQLONG   Version;         /* Structure version number */
    MQLONG   StrucLength;     /* Length of MQMDE structure */
    MQLONG   Encoding;        /* Numeric encoding of data that follows
                               MQMDE */
    MQLONG   CodedCharSetId; /* Character-set identifier of data that
                               follows MQMDE */
};
```

MQMDE – Language declarations

```
MQCHAR8  Format;          /* Format name of data that follows
                          MQMDE */
MQLONG   Flags;          /* General flags */
MQBYTE24 GroupId;       /* Group identifier */
MQLONG   MsgSeqNumber;  /* Sequence number of logical message
                          within group */
MQLONG   Offset;        /* Offset of data in physical message from
                          start of logical message */
MQLONG   MsgFlags;      /* Message flags */
MQLONG   OriginalLength; /* Length of original message */
};
```

COBOL declaration

```
** MQMDE structure
10 MQMDE.
** Structure identifier
15 MQMDE-STRUCID PIC X(4).
** Structure version number
15 MQMDE-VERSION PIC S9(9) BINARY.
** Length of MQMDE structure
15 MQMDE-STRUCLength PIC S9(9) BINARY.
** Numeric encoding of data that follows MQMDE
15 MQMDE-ENCODING PIC S9(9) BINARY.
** Character-set identifier of data that follows MQMDE
15 MQMDE-CODEDCHARSETID PIC S9(9) BINARY.
** Format name of data that follows MQMDE
15 MQMDE-FORMAT PIC X(8).
** General flags
15 MQMDE-FLAGS PIC S9(9) BINARY.
** Group identifier
15 MQMDE-GROUPID PIC X(24).
** Sequence number of logical message within group
15 MQMDE-MSGSEQNUMBER PIC S9(9) BINARY.
** Offset of data in physical message from start of logical message
15 MQMDE-OFFSET PIC S9(9) BINARY.
** Message flags
15 MQMDE-MSGFLAGS PIC S9(9) BINARY.
** Length of original message
15 MQMDE-ORIGINALLENGTH PIC S9(9) BINARY.
```

PL/I declaration

```
dc1
1 MQMDE based,
3 StrucId char(4), /* Structure identifier */
3 Version fixed bin(31), /* Structure version number */
3 StrucLength fixed bin(31), /* Length of MQMDE structure */
3 Encoding fixed bin(31), /* Numeric encoding of data that
                          follows MQMDE */
3 CodedCharSetId fixed bin(31), /* Character-set identifier of data
                          that follows MQMDE */
3 Format char(8), /* Format name of data that follows
                          MQMDE */
3 Flags fixed bin(31), /* General flags */
3 GroupId char(24), /* Group identifier */
3 MsgSeqNumber fixed bin(31), /* Sequence number of logical message
                          within group */
3 Offset fixed bin(31), /* Offset of data in physical message
                          from start of logical message */
3 MsgFlags fixed bin(31), /* Message flags */
3 OriginalLength fixed bin(31); /* Length of original message */
```

MQMDE – Language declarations

System/390 assembler declaration

```
MQMDE          DSECT
MQMDE_STRUCID  DS   CL4  Structure identifier
MQMDE_VERSION  DS   F    Structure version number
MQMDE_STRUCLNGTH DS   F    Length of MQMDE structure
MQMDE_ENCODING DS   F    Numeric encoding of data that follows
*              MQMDE
MQMDE_CODEDCHARSETID DS   F    Character-set identifier of data that
*              follows MQMDE
MQMDE_FORMAT   DS   CL8  Format name of data that follows MQMDE
MQMDE_FLAGS    DS   F    General flags
MQMDE_GROUPID  DS   XL24 Group identifier
MQMDE_MSGSEQNUMBER DS   F    Sequence number of logical message
*              within group
MQMDE_OFFSET   DS   F    Offset of data in physical message from
*              start of logical message
MQMDE_MSGFLAGS DS   F    Message flags
MQMDE_ORIGINALLENGTH DS   F    Length of original message
*
MQMDE_LENGTH   EQU   *-MQMDE
               ORG   MQMDE
MQMDE_AREA     DS   CL(MQMDE_LENGTH)
```

Visual Basic declaration

```
Type MQMDE
  StrucId      As String*4 'Structure identifier'
  Version      As Long     'Structure version number'
  StrucLength  As Long     'Length of MQMDE structure'
  Encoding     As Long     'Numeric encoding of data that follows'
                'MQMDE'
  CodedCharSetId As Long   'Character-set identifier of data that'
                'follows MQMDE'
  Format       As String*8 'Format name of data that follows MQMDE'
  Flags        As Long     'General flags'
  GroupId      As MQBYTE24 'Group identifier'
  MsgSeqNumber As Long     'Sequence number of logical message within'
                'group'
  Offset       As Long     'Offset of data in physical message from'
                'start of logical message'
  MsgFlags     As Long     'Message flags'
  OriginalLength As Long   'Length of original message'
End Type
```

Chapter 12. MQOD – Object descriptor

The following table summarizes the fields in the structure.

Table 56. Fields in MQOD

Field	Description	Page
<i>StrucId</i>	Structure identifier	221
<i>Version</i>	Structure version number	221
<i>ObjectType</i>	Object type	218
<i>ObjectName</i>	Object name	215
<i>ObjectQMgrName</i>	Object queue manager name	216
<i>DynamicQName</i>	Dynamic queue name	214
<i>AlternateUserId</i>	Alternate user identifier	213
Note: The remaining fields are ignored if <i>Version</i> is less than MQOD_VERSION_2.		
<i>RecsPresent</i>	Number of object records present	218
<i>KnownDestCount</i>	Number of local queues opened successfully	215
<i>UnknownDestCount</i>	Number of remote queues opened successfully	221
<i>InvalidDestCount</i>	Number of queues that failed to open	214
<i>ObjectRecOffset</i>	Offset of first object record from start of MQOD	217
<i>ResponseRecOffset</i>	Offset of first response record from start of MQOD	220
<i>ObjectRecPtr</i>	Address of first object record	218
<i>ResponseRecPtr</i>	Address of first response record	220
Note: The remaining fields are ignored if <i>Version</i> is less than MQOD_VERSION_3.		
<i>AlternateSecurityId</i>	Alternate security identifier	212
<i>ResolvedQName</i>	Resolved queue name	219
<i>ResolvedQMgrName</i>	Resolved queue manager name	219

Overview

Availability:

- Version 1: All
- Versions 2 and 3: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems

Purpose: The MQOD structure is used to specify an object by name. The following types of object are valid:

- Queue or distribution list
- Namelist
- Process definition
- Queue manager

The structure is an input/output parameter on the MQOPEN and MQPUT1 calls.

MQOD – Object descriptor

Version: The current version of MQOD is MQOD_VERSION_3, but this version is not supported in all environments (see above). Applications that are intended to be portable between several environments must ensure that the required version of MQOD is supported in all of the environments concerned. Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions that follow.

The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQOD that is supported by the environment, but with the initial value of the *Version* field set to MQOD_VERSION_1. To use fields that are not present in the version-1 structure, the application must set the *Version* field to the version number of the version required.

To open a distribution list, *Version* must be MQOD_VERSION_2 or greater.

Character set and encoding: Data in MQOD must be in the character set and encoding of the local queue manager; these are given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE, respectively. However, if the application is running as an MQ client, the structure must be in the character set and encoding of the client.

Fields

The MQOD structure contains the following fields; the fields are described in **alphabetic order**:

AlternateSecurityId (MQBYTE40)

Alternate security identifier.

This is a security identifier that is passed with the *AlternateUserId* to the authorization service to allow appropriate authorization checks to be performed. *AlternateSecurityId* is used only if:

- MQOO_ALTERNATE_USER_AUTHORITY is specified on the MQOPEN call, or
- MQPMO_ALTERNATE_USER_AUTHORITY is specified on the MQPUT1 call, and the *AlternateUserId* field is not entirely blank up to the first null character or the end of the field.

On Windows, *AlternateSecurityId* can be used to supply the Windows NT security identifier (SID) that uniquely identifies the *AlternateUserId*. The SID for a user can be obtained from the Windows NT system by use of the `LookupAccountName()` Windows API call.

On z/OS, this field is ignored.

The *AlternateSecurityId* field has the following structure:

- The first byte is a binary integer containing the length of the significant data that follows; the value excludes the length byte itself. If no security identifier is present, the length is zero.
- The second byte indicates the type of security identifier that is present; the following values are possible:
MQSIDT_NT_SECURITY_ID
Windows security identifier.

MQSIDT_NONE

No security identifier.

- The third and subsequent bytes up to the length defined by the first byte contain the security identifier itself.
- Remaining bytes in the field are set to binary zero.

The following special value may be used:

MQSID_NONE

No security identifier specified.

The value is binary zero for the length of the field.

For the C programming language, the constant `MQSID_NONE_ARRAY` is also defined; this has the same value as `MQSID_NONE`, but is an array of characters instead of a string.

This is an input field. The length of this field is given by `MQ_SECURITY_ID_LENGTH`. The initial value of this field is `MQSID_NONE`. This field is ignored if *Version* is less than `MQOD_VERSION_3`.

AlternateUserId (MQCHAR12)

Alternate user identifier.

If `MQOO_ALTERNATE_USER_AUTHORITY` is specified for the `MQOPEN` call, or `MQPMO_ALTERNATE_USER_AUTHORITY` for the `MQPUT1` call, this field contains an alternate user identifier that is to be used to check the authorization for the open, in place of the user identifier that the application is currently running under. Some checks, however, are still carried out with the current user identifier (for example, context checks).

If `MQOO_ALTERNATE_USER_AUTHORITY` or `MQPMO_ALTERNATE_USER_AUTHORITY` is specified and this field is entirely blank up to the first null character or the end of the field, the open can succeed only if no user authorization is needed to open this object with the options specified.

If neither `MQOO_ALTERNATE_USER_AUTHORITY` nor `MQPMO_ALTERNATE_USER_AUTHORITY` is specified, this field is ignored.

The following differences exist in the environments indicated:

- On Windows 3.1, Windows 95, Windows 98, the value in this field is accepted but ignored.
- On z/OS, only the first 8 characters of *AlternateUserId* are used to check the authorization for the open. However, the current user identifier must be authorized to specify this particular alternate user identifier; all 12 characters of the alternate user identifier are used for this check. The user identifier must contain only characters allowed by the external security manager.

If *AlternateUserId* is specified for a queue, the value may be used subsequently by the queue manager when messages are put. If the `MQPMO_*_CONTEXT` options specified on the `MQPUT` or `MQPUT1` call cause the queue manager to generate the identity context information, the queue manager places the *AlternateUserId* into the *UserIdentifier* field in the `MQMD` of the message, in place of the current user identifier.

MQOD – AlternateUserId field

- In other environments, *AlternateUserId* is used only for access control checks on the object being opened. If the object is a queue, *AlternateUserId* does not affect the content of the *UserIdentifier* field in the MQMD of messages sent using that queue handle.

This is an input field. The length of this field is given by MQ_USER_ID_LENGTH. The initial value of this field is the null string in C, and 12 blank characters in other programming languages.

DynamicQName (MQCHAR48)

Dynamic queue name.

This is the name of a dynamic queue that is to be created by the MQOPEN call. This is of relevance only when *ObjectName* specifies the name of a model queue; in all other cases *DynamicQName* is ignored.

The characters that are valid in the name are the same as those for *ObjectName* (see above), except that an asterisk is also valid (see below). A name that is completely blank (or one in which only blanks appear before the first null character) is not valid if *ObjectName* is the name of a model queue.

If the last nonblank character in the name is an asterisk (*), the queue manager replaces the asterisk with a string of characters that guarantees that the name generated for the queue is unique at the local queue manager. To allow a sufficient number of characters for this, the asterisk is valid only in positions 1 through 33. There must be no characters other than blanks or a null character following the asterisk.

It is valid for the asterisk to appear in the first character position, in which case the name consists solely of the characters generated by the queue manager.

On z/OS, it is not recommended to use a name with the asterisk in the first character position, as there can be no security checks made on a queue whose full name is generated automatically.

This is an input field. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is determined by the environment:

- On z/OS, the value is 'CSQ.*'.
- On other platforms, the value is 'AMQ.*'.

The value is a null-terminated string in C, and a blank-padded string in other programming languages.

InvalidDestCount (MQLONG)

Number of queues that failed to open.

This is the number of queues in the distribution list that failed to open successfully. If present, this field is also set when opening a single queue which is not in a distribution list.

Note: If present, this field is set *only* if the *CompCode* parameter on the MQOPEN or MQPUT1 call is MQCC_OK or MQCC_WARNING; it is *not* set if the *CompCode* parameter is MQCC_FAILED.

This is an output field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_2.

KnownDestCount (MQLONG)

Number of local queues opened successfully.

This is the number of queues in the distribution list that resolve to local queues and that were opened successfully. The count does not include queues that resolve to remote queues (even though a local transmission queue is used initially to store the message). If present, this field is also set when opening a single queue which is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_2.

ObjectName (MQCHAR48)

Object name.

This is the local name of the object as defined on the queue manager identified by *ObjectQMGrName*. The name can contain the following characters:

- Uppercase alphabetic characters (A through Z)
- Lowercase alphabetic characters (a through z)
- Numeric digits (0 through 9)
- Period (.), forward slash (/), underscore (_), percent (%)

The name must not contain leading or embedded blanks, but may contain trailing blanks. A null character can be used to indicate the end of significant data in the name; the null and any characters following it are treated as blanks. The following restrictions apply in the environments indicated:

- On systems that use EBCDIC Katakana, lowercase characters cannot be used.
- On z/OS:
 - Names that begin or end with an underscore cannot be processed by the operations and control panels. For this reason such names should be avoided.
 - The percent character has a special meaning to RACF. If RACF is used as the external security manager, names should not contain the percent. If they do, those names are not included in any security checks when RACF generic profiles are used.
- On OS/400, names containing lowercase characters, forward slash, or percent, must be enclosed in quotation marks when specified on commands. These quotation marks must not be specified for names that occur as fields in structures or as parameters on calls.

The following points apply to the types of object indicated:

- If *ObjectName* is the name of a model queue, the queue manager creates a dynamic queue with the attributes of the model queue, and returns in the *ObjectName* field the name of the queue created. A model queue can be specified only on the MQOPEN call; a model queue is not valid on the MQPUT1 call.
- If *ObjectName* and *ObjectQMGrName* identify a shared queue owned by the queue-sharing group to which the local queue manager belongs, there must not also be a queue definition of the same name on the local queue manager. If there is such a definition (a local queue, alias queue, remote queue, or model queue), the call fails with reason code MQRC_OBJECT_NOT_UNIQUE.

MQOD – ObjectName field

- If the object being opened is a distribution list (that is, *RecsPresent* is present and greater than zero), *ObjectName* must be blank or the null string. If this condition is not satisfied, the call fails with reason code MQRC_OBJECT_NAME_ERROR.
- If *ObjectType* is MQOT_Q_MGR, special rules apply; in this case the name must be entirely blank up to the first null character or the end of the field.

This is an input/output field for the MQOPEN call when *ObjectName* is the name of a model queue, and an input-only field in all other cases. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ObjectQMgrName (MQCHAR48)

Object queue manager name.

This is the name of the queue manager on which the *ObjectName* object is defined. The characters that are valid in the name are the same as those for *ObjectName* (see above). A name that is entirely blank up to the first null character or the end of the field denotes the queue manager to which the application is connected (the local queue manager).

The following points apply to the types of object indicated:

- If *ObjectType* is MQOT_NAMELIST, MQOT_PROCESS, or MQOT_Q_MGR, *ObjectQMgrName* must be blank or the name of the local queue manager.
- If *ObjectName* is the name of a model queue, the queue manager creates a dynamic queue with the attributes of the model queue, and returns in the *ObjectQMgrName* field the name of the queue manager on which the queue is created; this is the name of the local queue manager. A model queue can be specified only on the MQOPEN call; a model queue is not valid on the MQPUT1 call.
- If *ObjectName* is the name of a cluster queue, and *ObjectQMgrName* is blank, the actual destination of messages sent using the queue handle returned by the MQOPEN call is chosen by the queue manager (or cluster workload exit, if one is installed) as follows:
 - If MQOO_BIND_ON_OPEN is specified, the queue manager selects a particular instance of the cluster queue during the processing of the MQOPEN call, and all messages put using this queue handle are sent to that instance.
 - If MQOO_BIND_NOT_FIXED is specified, the queue manager may choose a different instance of the destination queue (residing on a different queue manager in the cluster) for each successive MQPUT call that uses this queue handle.

If the application needs to send a message to a *specific* instance of a cluster queue (that is, a queue instance that resides on a particular queue manager in the cluster), the application should specify the name of that queue manager in the *ObjectQMgrName* field. This forces the local queue manager to send the message to the specified destination queue manager.

- If *ObjectName* is the name of a shared queue that is owned by the queue-sharing group to which the local queue manager belongs, *ObjectQMgrName* can be the name of the queue-sharing group, the name of the local queue manager, or blank; the message is placed on the same queue whichever of these values is specified.

Queue-sharing groups are supported only on z/OS.

- If *ObjectName* is the name of a shared queue that is owned by a remote queue-sharing group (that is, a queue-sharing group to which the local queue manger does *not* belong), *ObjectQMgrName* should be the name of the queue-sharing group. The name of a queue manager that belongs to that group is also valid, but this is not recommended as it may cause the message to be delayed if that particular queue manager is not available when the message arrives at the queue-sharing group.
- If the object being opened is a distribution list (that is, *RecsPresent* is greater than zero), *ObjectQMgrName* must be blank or the null string. If this condition is not satisfied, the call fails with reason code MQRC_OBJECT_Q_MGR_NAME_ERROR.

This is an input/output field for the MQOPEN call when *ObjectName* is the name of a model queue, and an input-only field in all other cases. The length of this field is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ObjectRecOffset (MQLONG)

Offset of first object record from start of MQOD.

This is the offset in bytes of the first MQOR object record from the start of the MQOD structure. The offset can be positive or negative. *ObjectRecOffset* is used only when a distribution list is being opened. The field is ignored if *RecsPresent* is zero.

When a distribution list is being opened, an array of one or more MQOR object records must be provided in order to specify the names of the destination queues in the distribution list. This can be done in one of two ways:

- By using the offset field *ObjectRecOffset*

In this case, the application should declare its own structure containing an MQOD followed by the array of MQOR records (with as many array elements as are needed), and set *ObjectRecOffset* to the offset of the first element in the array from the start of the MQOD. Care must be taken to ensure that this offset is correct and has a value that can be accommodated within an MQLONG (the most restrictive programming language is COBOL, for which the valid range is –999 999 999 through +999 999 999).

Using *ObjectRecOffset* is recommended for programming languages which do not support the pointer data type, or which implement the pointer data type in a fashion which is not portable to different environments (for example, the COBOL programming language).

- By using the pointer field *ObjectRecPtr*

In this case, the application can declare the array of MQOR structures separately from the MQOD structure, and set *ObjectRecPtr* to the address of the array.

Using *ObjectRecPtr* is recommended for programming languages which support the pointer data type in a fashion which is portable to different environments (for example, the C programming language).

Whichever technique is chosen, one of *ObjectRecOffset* and *ObjectRecPtr* must be used; the call fails with reason code MQRC_OBJECT_RECORDS_ERROR if both are zero, or both are nonzero.

MQOD – ObjectRecOffset field

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_2.

ObjectRecPtr (MQPTR)

Address of first object record.

This is the address of the first MQOR object record. *ObjectRecPtr* is used only when a distribution list is being opened. The field is ignored if *RecsPresent* is zero.

Either *ObjectRecPtr* or *ObjectRecOffset* can be used to specify the object records, but not both; see the description of the *ObjectRecOffset* field above for details. If *ObjectRecPtr* is not used, it must be set to the null pointer or null bytes.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQOD_VERSION_2.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

ObjectType (MQLONG)

Object type.

Type of object being named in *ObjectName*. Possible values are:

MQOT_Q

Queue.

MQOT_NAMELIST

Namelist.

This is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

MQOT_PROCESS

Process definition.

This is not supported in the following environments: VSE/ESA, Windows 3.1, Windows 95, Windows 98.

MQOT_Q_MGR

Queue manager.

This is not supported on VSE/ESA.

This is always an input field. The initial value of this field is MQOT_Q.

RecsPresent (MQLONG)

Number of object records present.

This is the number of MQOR object records that have been provided by the application. If this number is greater than zero, it indicates that a distribution list is being opened, with *RecsPresent* being the number of destination queues in the list. It is valid for a distribution list to contain only one destination.

The value of *RecsPresent* must not be less than zero, and if it is greater than zero *ObjectType* must be MQOT_Q; the call fails with reason code MQRC_RECS_PRESENT_ERROR if these conditions are not satisfied.

On z/OS, this field must be zero.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_2.

ResolvedQMgrName (MQCHAR48)

Resolved queue manager name.

This is the name of the destination queue manager after name resolution has been performed by the local queue manager. The name returned is the name of the queue manager that owns the queue identified by *ResolvedQName*. *ResolvedQMgrName* can be the name of the local queue manager.

If *ResolvedQName* is a shared queue that is owned by the queue-sharing group to which the local queue manager belongs, *ResolvedQMgrName* is the name of the queue-sharing group. If the queue is owned by some other queue-sharing group, *ResolvedQName* can be the name of the queue-sharing group or the name of a queue manager that is a member of the queue-sharing group (the nature of the value returned is determined by the queue definitions that exist at the local queue manager).

A nonblank value is returned only if the object is a single queue opened for browse, input, or output (or any combination). If the object opened is any of the following, *ResolvedQMgrName* is set to blanks:

- Not a queue
- A queue, but not opened for browse, input, or output
- A cluster queue with MQOO_BIND_NOT_FIXED specified (or with MQOO_BIND_AS_Q_DEF in effect when the *DefBind* queue attribute has the value MQBND_BIND_NOT_FIXED)
- A distribution list

This is an output field. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages. This field is ignored if *Version* is less than MQOD_VERSION_3.

ResolvedQName (MQCHAR48)

Resolved queue name.

This is the name of the destination queue after name resolution has been performed by the local queue manager. The name returned is the name of a queue that exists on the queue manager identified by *ResolvedQMgrName*.

A nonblank value is returned only if the object is a single queue opened for browse, input, or output (or any combination). If the object opened is any of the following, *ResolvedQName* is set to blanks:

- Not a queue
- A queue, but not opened for browse, input, or output
- A distribution list

MQOD – ResolvedQName field

This is an output field. The length of this field is given by `MQ_Q_NAME_LENGTH`. The initial value of this field is the null string in C, and 48 blank characters in other programming languages. This field is ignored if *Version* is less than `MQOD_VERSION_3`.

ResponseRecOffset (MQLONG)

Offset of first response record from start of MQOD.

This is the offset in bytes of the first MQRR response record from the start of the MQOD structure. The offset can be positive or negative. *ResponseRecOffset* is used only when a distribution list is being opened. The field is ignored if *RecsPresent* is zero.

When a distribution list is being opened, an array of one or more MQRR response records can be provided in order to identify the queues that failed to open (*CompCode* field in MQRR), and the reason for each failure (*Reason* field in MQRR). The data is returned in the array of response records in the same order as the queue names occur in the array of object records. The queue manager sets the response records only when the outcome of the call is mixed (that is, some queues were opened successfully while others failed, or all failed but for differing reasons); reason code `MQRC_MULTIPLE_REASONS` from the call indicates this case. If the same reason code applies to all queues, that reason is returned in the *Reason* parameter of the MQOPEN or MQPUT1 call, and the response records are not set. Response records are optional, but if they are supplied there must be *RecsPresent* of them.

The response records can be provided in the same way as the object records, either by specifying an offset in *ResponseRecOffset*, or by specifying an address in *ResponseRecPtr*; see the description of *ObjectRecOffset* above for details of how to do this. However, no more than one of *ResponseRecOffset* and *ResponseRecPtr* can be used; the call fails with reason code `MQRC_RESPONSE_RECORDS_ERROR` if both are nonzero.

For the MQPUT1 call, these response records are used to return information about errors that occur when the message is sent to the queues in the distribution list, as well as errors that occur when the queues are opened. The completion code and reason code from the put operation for a queue replace those from the open operation for that queue only if the completion code from the latter was `MQCC_OK` or `MQCC_WARNING`.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than `MQOD_VERSION_2`.

ResponseRecPtr (MQPTR)

Address of first response record.

This is the address of the first MQRR response record. *ResponseRecPtr* is used only when a distribution list is being opened. The field is ignored if *RecsPresent* is zero.

Either *ResponseRecPtr* or *ResponseRecOffset* can be used to specify the response records, but not both; see the description of the *ResponseRecOffset* field above for details. If *ResponseRecPtr* is not used, it must be set to the null pointer or null bytes.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQOD_VERSION_2.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQOD_STRUC_ID

Identifier for object descriptor structure.

For the C programming language, the constant MQOD_STRUC_ID_ARRAY is also defined; this has the same value as MQOD_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQOD_STRUC_ID.

UnknownDestCount (MQLONG)

Number of remote queues opened successfully

This is the number of queues in the distribution list that resolve to remote queues and that were opened successfully. If present, this field is also set when opening a single queue which is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_2.

Version (MQLONG)

Structure version number.

The value must be one of the following:

MQOD_VERSION_1

Version-1 object descriptor structure.

This version is supported in all environments.

MQOD_VERSION_2

Version-2 object descriptor structure.

This version is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

MQOD_VERSION_3

Version-3 object descriptor structure.

This version is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

MQOD – Version field

Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQOD_CURRENT_VERSION

Current version of object descriptor structure.

This is always an input field. The initial value of this field is MQOD_VERSION_1.

Initial values and language declarations

Table 57. Initial values of fields in MQOD

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQOD_STRUC_ID	'0Dbb'
<i>Version</i>	MQOD_VERSION_1	1
<i>ObjectType</i>	MQOT_Q	1
<i>ObjectName</i>	None	Null string or blanks
<i>ObjectQMgrName</i>	None	Null string or blanks
<i>DynamicQName</i>	None	'CSQ.*' on z/OS; 'AMQ.*' otherwise
<i>AlternateUserId</i>	None	Null string or blanks
<i>RecsPresent</i>	None	0
<i>KnownDestCount</i>	None	0
<i>UnknownDestCount</i>	None	0
<i>InvalidDestCount</i>	None	0
<i>ObjectRecOffset</i>	None	0
<i>ResponseRecOffset</i>	None	0
<i>ObjectRecPtr</i>	None	Null pointer or null bytes
<i>ResponseRecPtr</i>	None	Null pointer or null bytes
<i>AlternateSecurityId</i>	MQSID_NONE	Nulls
<i>ResolvedQName</i>	None	Null string or blanks
<i>ResolvedQMgrName</i>	None	Null string or blanks

Notes:

1. The symbol 'b' represents a single blank character.
2. The value 'Null string or blanks' denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQOD_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:
MQOD MyOD = {MQOD_DEFAULT};

C declaration

```
typedef struct tagMQOD MQOD;  
struct tagMQOD {  
    MQCHAR4    StrucId;           /* Structure identifier */  
    MQLONG     Version;          /* Structure version number */  
    MQLONG     ObjectType;       /* Object type */
```

MQOD – Language declarations

```
MQCHAR48 ObjectName;          /* Object name */
MQCHAR48 ObjectQMgrName;     /* Object queue manager name */
MQCHAR48 DynamicQName;      /* Dynamic queue name */
MQCHAR12 AlternateUserId;   /* Alternate user identifier */
MQLONG   RecsPresent;       /* Number of object records present */
MQLONG   KnownDestCount;    /* Number of local queues opened
                             successfully */
MQLONG   UnknownDestCount;  /* Number of remote queues opened
                             successfully */
MQLONG   InvalidDestCount;  /* Number of queues that failed to
                             open */
MQLONG   ObjectRecOffset;   /* Offset of first object record from
                             start of MQOD */
MQLONG   ResponseRecOffset; /* Offset of first response record
                             from start of MQOD */

MQPTR    ObjectRecPtr;      /* Address of first object record */
MQPTR    ResponseRecPtr;    /* Address of first response record */
MQBYTE40 AlternateSecurityId; /* Alternate security identifier */
MQCHAR48 ResolvedQName;     /* Resolved queue name */
MQCHAR48 ResolvedQMgrName;  /* Resolved queue manager name */
};
```

COBOL declaration

```
** MQOD structure
10 MQOD.
** Structure identifier
15 MQOD-STRUCID          PIC X(4).
** Structure version number
15 MQOD-VERSION         PIC S9(9) BINARY.
** Object type
15 MQOD-OBJECTTYPE     PIC S9(9) BINARY.
** Object name
15 MQOD-OBJECTNAME     PIC X(48).
** Object queue manager name
15 MQOD-OBJECTQMGRNAME PIC X(48).
** Dynamic queue name
15 MQOD-DYNAMICQNAME   PIC X(48).
** Alternate user identifier
15 MQOD-ALTERNATEUSERID PIC X(12).
** Number of object records present
15 MQOD-RECSPRESENT    PIC S9(9) BINARY.
** Number of local queues opened successfully
15 MQOD-KNOWNDSTCOUNT PIC S9(9) BINARY.
** Number of remote queues opened successfully
15 MQOD-UNKNOWNDSTCOUNT PIC S9(9) BINARY.
** Number of queues that failed to open
15 MQOD-INVALIDDSTCOUNT PIC S9(9) BINARY.
** Offset of first object record from start of MQOD
15 MQOD-OBJECTRECOFFSET PIC S9(9) BINARY.
** Offset of first response record from start of MQOD
15 MQOD-RESPONSERECOFFSET PIC S9(9) BINARY.
** Address of first object record
15 MQOD-OBJECTRECPTTR  POINTER.
** Address of first response record
15 MQOD-RESPONSERECPTTR POINTER.
** Alternate security identifier
15 MQOD-ALTERNATESECURITYID PIC X(40).
** Resolved queue name
15 MQOD-RESOLVEDQNAME  PIC X(48).
** Resolved queue manager name
15 MQOD-RESOLVEDQMGRNAME PIC X(48).
```

MQOD – Language declarations

PL/I declaration

```
dc1
1 MQOD based,
3 StrucId          char(4),          /* Structure identifier */
3 Version         fixed bin(31), /* Structure version number */
3 ObjectType      fixed bin(31), /* Object type */
3 ObjectName      char(48),        /* Object name */
3 ObjectQMgrName  char(48),        /* Object queue manager name */
3 DynamicQName    char(48),        /* Dynamic queue name */
3 AlternateUserId char(12),        /* Alternate user identifier */
3 RecsPresent     fixed bin(31), /* Number of object records
                                present */
3 KnownDestCount  fixed bin(31), /* Number of local queues opened
                                successfully */
3 UnknownDestCount fixed bin(31), /* Number of remote queues opened
                                successfully */
3 InvalidDestCount fixed bin(31), /* Number of queues that failed
                                to open */
3 ObjectRecOffset fixed bin(31), /* Offset of first object record
                                from start of MQOD */
3 ResponseRecOffset fixed bin(31), /* Offset of first response
                                record from start of MQOD */
3 ObjectRecPtr    pointer,         /* Address of first object
                                record */
3 ResponseRecPtr  pointer,         /* Address of first response
                                record */
3 AlternateSecurityId char(40),    /* Alternate security
                                identifier */
3 ResolvedQName   char(48),        /* Resolved queue name */
3 ResolvedQMgrName char(48);      /* Resolved queue manager name */
```

System/390 assembler declaration

```
MQOD          DSECT
MQOD_STRUCID  DS   CL4   Structure identifier
MQOD_VERSION  DS   F     Structure version number
MQOD_OBJECTTYPE DS   F     Object type
MQOD_OBJECTNAME DS  CL48  Object name
MQOD_OBJECTQMGRNAME DS  CL48  Object queue manager name
MQOD_DYNAMICQNAME DS  CL48  Dynamic queue name
MQOD_ALTERNATEUSERID DS  CL12  Alternate user identifier
MQOD_RECSPRESENT DS   F     Number of object records present
MQOD_KNOWNDSTCOUNT DS   F     Number of local queues opened
*                                successfully
MQOD_UNKNOWNDSTCOUNT DS   F     Number of remote queues opened
*                                successfully
MQOD_INVALIDDSTCOUNT DS   F     Number of queues that failed to
*                                open
MQOD_OBJECTRECOFFSET DS   F     Offset of first object record from
*                                start of MQOD
MQOD_RESPONSERECOFFSET DS   F     Offset of first response record
*                                from start of MQOD
MQOD_OBJECTRECPTTR DS   F     Address of first object record
MQOD_RESPONSERECPTTR DS   F     Address of first response record
MQOD_ALTERNATESECURITYID DS  XL40  Alternate security identifier
MQOD_RESOLVEDQNAME DS  CL48  Resolved queue name
MQOD_RESOLVEDQMGRNAME DS  CL48  Resolved queue manager name
*
MQOD_LENGTH   EQU  *-MQOD
ORG  MQOD
MQOD_AREA     DS   CL(MQOD_LENGTH)
```

TAL declaration

```

STRUCT      MQOD^DEF (*);BEGINSTRUCT      STRUCID;
BEGIN STRING BYTE [0:3]; END;INT(32)      VERSION;
INT(32)     OBJECTTYPE;STRUCT
          OBJECTNAME;
BEGIN STRING BYTE [0:47]; END;STRUCT      OBJECTQMGRNAME;
BEGIN STRING BYTE [0:47]; END;STRUCT      DYNAMICQNAME;
BEGIN STRING BYTE [0:47]; END;STRUCT      ALTERNATEUSERID;
BEGIN STRING BYTE [0:11]; END;

```

Visual Basic declaration

```

Type MQOD
  StrucId      As String*4  'Structure identifier'
  Version      As Long      'Structure version number'
  ObjectType   As Long      'Object type'
  ObjectName   As String*48 'Object name'
  ObjectQMgrName As String*48 'Object queue manager name'
  DynamicQName As String*48 'Dynamic queue name'
  AlternateUserId As String*12 'Alternate user identifier'
  RecsPresent  As Long      'Number of object records present'
  KnownDestCount As Long      'Number of local queues opened'
                                'successfully'
  UnknownDestCount As Long      'Number of remote queues opened'
                                'successfully'
  InvalidDestCount As Long      'Number of queues that failed to'
                                'open'
  ObjectRecOffset As Long      'Offset of first object record from'
                                'start of MQOD'
  ResponseRecOffset As Long      'Offset of first response record'
                                'from start of MQOD'
  ObjectRecPtr  As MQPTR      'Address of first object record'
  ResponseRecPtr As MQPTR      'Address of first response record'
  AlternateSecurityId As MQBYTE40 'Alternate security identifier'
  ResolvedQName  As String*48 'Resolved queue name'
  ResolvedQMgrName As String*48 'Resolved queue manager name'
End Type

```

MQOD – Language declarations

Chapter 13. MQOR – Object record

The following table summarizes the fields in the structure.

Table 58. Fields in MQOR

Field	Description	Page
<i>ObjectName</i>	Object name	227
<i>ObjectQMgrName</i>	Object queue manager name	227

Overview

Availability: AIX, HP-UX, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Purpose: The MQOR structure is used to specify the queue name and queue-manager name of a single destination queue. MQOR is an input structure for the MQOPEN and MQPUT1 calls.

Character set and encoding: Data in MQOR must be in the character set and encoding of the local queue manager; these are given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE, respectively. However, if the application is running as an MQ client, the structure must be in the character set and encoding of the client.

Usage: By providing an array of these structures on the MQOPEN call, it is possible to open a list of queues; this list is called a *distribution list*. Each message put using the queue handle returned by that MQOPEN call is placed on each of the queues in the list, provided that the queue was opened successfully.

Fields

The MQOR structure contains the following fields; the fields are described in **alphabetic order**:

ObjectName (MQCHAR48)

Object name.

This is the same as the *ObjectName* field in the MQOD structure (see MQOD for details), except that:

- It must be the name of a queue.
- It must not be the name of a model queue.

This is always an input field. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ObjectQMgrName (MQCHAR48)

Object queue manager name.

MQOR – ObjectQMgrName field

This is the same as the *ObjectQMgrName* field in the MQOD structure (see MQOD for details).

This is always an input field. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

Initial values and language declarations

Table 59. Initial values of fields in MQOR

Field name	Name of constant	Value of constant
<i>ObjectName</i>	None	Null string or blanks
<i>ObjectQMgrName</i>	None	Null string or blanks

Notes:

1. The value 'Null string or blanks' denotes the null string in C, and blank characters in other programming languages.
2. In the C programming language, the macro variable MQOR_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:
MQOR MyOR = {MQOR_DEFAULT};

C declaration

```
typedef struct tagMQOR MQOR;  
struct tagMQOR {  
    MQCHAR48 ObjectName;    /* Object name */  
    MQCHAR48 ObjectQMgrName; /* Object queue manager name */  
};
```

COBOL declaration

```
** MQOR structure  
10 MQOR.  
**   Object name  
15 MQOR-OBJECTNAME PIC X(48).  
**   Object queue manager name  
15 MQOR-OBJECTQMGRNAME PIC X(48).
```

PL/I declaration

```
dcl  
1 MQOR based,  
3 ObjectName char(48), /* Object name */  
3 ObjectQMgrName char(48); /* Object queue manager name */
```

Visual Basic declaration

```
Type MQOR  
    ObjectName As String*48 'Object name'  
    ObjectQMgrName As String*48 'Object queue manager name'  
End Type
```

Chapter 14. MQPMO – Put-message options

The following table summarizes the fields in the structure.

Table 60. Fields in MQPMO

Field	Description	Page
<i>StrucId</i>	Structure identifier	245
<i>Version</i>	Structure version number	245
<i>Options</i>	Options that control the action of MQPUT and MQPUT1	231
<i>Timeout</i>	Reserved	245
<i>Context</i>	Object handle of input queue	230
<i>KnownDestCount</i>	Number of messages sent successfully to local queues	230
<i>UnknownDestCount</i>	Number of messages sent successfully to remote queues	245
<i>InvalidDestCount</i>	Number of messages that could not be sent	230
<i>ResolvedQName</i>	Resolved name of destination queue	243
<i>ResolvedQMGrName</i>	Resolved name of destination queue manager	243
Note: The remaining fields are ignored if <i>Version</i> is less than MQPMO_VERSION_2.		
<i>RecsPresent</i>	Number of put message records or response records present	242
<i>PutMsgRecFields</i>	Flags indicating which MQPMR fields are present	240
<i>PutMsgRecOffset</i>	Offset of first put-message record from start of MQPMO	241
<i>ResponseRecOffset</i>	Offset of first response record from start of MQPMO	243
<i>PutMsgRecPtr</i>	Address of first put message record	242
<i>ResponseRecPtr</i>	Address of first response record	244

Overview

Availability:

- Version 1: All
- Version 2: AIX, HP-UX, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems

Purpose: The MQPMO structure allows the application to specify options that control how messages are placed on queues. The structure is an input/output parameter on the MQPUT and MQPUT1 calls.

Version: The current version of MQPMO is MQPMO_VERSION_2, but this version is not supported in all environments (see above). Applications that are intended to be portable between several environments must ensure that the required version of

MQPMO – Put-message options

MQPMO is supported in all of the environments concerned. Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions that follow.

The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQPMO that is supported by the environment, but with the initial value of the *Version* field set to MQPMO_VERSION_1. To use fields that are not present in the version-1 structure, the application must set the *Version* field to the version number of the version required.

Character set and encoding: Data in MQPMO must be in the character set and encoding of the local queue manager; these are given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE, respectively. However, if the application is running as an MQ client, the structure must be in the character set and encoding of the client.

Fields

The MQPMO structure contains the following fields; the fields are described in **alphabetic order**:

Context (MQHOBJ)

Object handle of input queue.

If MQPMO_PASS_IDENTITY_CONTEXT or MQPMO_PASS_ALL_CONTEXT is specified, this field must contain the input queue handle from which context information to be associated with the message being put is taken.

If neither MQPMO_PASS_IDENTITY_CONTEXT nor MQPMO_PASS_ALL_CONTEXT is specified, this field is ignored.

This is an input field. The initial value of this field is 0.

InvalidDestCount (MQLONG)

Number of messages that could not be sent.

This is the number of messages that could not be sent to queues in the distribution list. The count includes queues that failed to open, as well as queues that were opened successfully but for which the put operation failed. This field is also set when putting a message to a single queue which is not in a distribution list.

Note: This field is set *only* if the *CompCode* parameter on the MQPUT or MQPUT1 call is MQCC_OK or MQCC_WARNING; it is *not* set if the *CompCode* parameter is MQCC_FAILED.

This is an output field. The initial value of this field is 0. This field is not set if *Version* is less than MQPMO_VERSION_2.

KnownDestCount (MQLONG)

Number of messages sent successfully to local queues.

This is the number of messages that the current MQPUT or MQPUT1 call has sent successfully to queues in the distribution list that are local queues. The count does

not include messages sent to queues that resolve to remote queues (even though a local transmission queue is used initially to store the message). This field is also set when putting a message to a single queue which is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is not set if *Version* is less than MQPMO_VERSION_2.

Options (MQLONG)

Options that control the action of MQPUT and MQPUT1.

Any or none of the following can be specified. If more than one is required the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

Combinations that are not valid are noted; any other combinations are valid.

Syncpoint options: The following options relate to the participation of the MQPUT or MQPUT1 call within a unit of work:

MQPMO_SYNCPOINT

Put message with syncpoint control.

The request is to operate within the normal unit-of-work protocols. The message is not visible outside the unit of work until the unit of work is committed. If the unit of work is backed out, the message is deleted.

If neither this option nor MQPMO_NO_SYNCPOINT is specified, the inclusion of the put request in unit-of-work protocols is determined by the environment:

- On z/OS, Compaq NonStop Kernel, and VSE/ESA, the put request is within a unit of work.
- In all other environments, the put request is not within a unit of work.

Because of these differences, an application which is intended to be portable should not allow this option to default; either MQPMO_SYNCPOINT or MQPMO_NO_SYNCPOINT should be specified explicitly.

MQPMO_SYNCPOINT must *not* be specified with MQPMO_NO_SYNCPOINT.

MQPMO_NO_SYNCPOINT

Put message without syncpoint control.

The request is to operate outside the normal unit-of-work protocols. The message is available immediately, and it cannot be deleted by backing out a unit of work.

If neither this option nor MQPMO_SYNCPOINT is specified, the inclusion of the put request in unit-of-work protocols is determined by the environment:

- On z/OS Compaq NonStop Kernel, and VSE/ESA, the put request is within a unit of work.
- In all other environments, the put request is not within a unit of work.

MQPMO – Options field

Because of these differences, an application which is intended to be portable should not allow this option to default; either MQPMO_SYNCPOINT or MQPMO_NO_SYNCPOINT should be specified explicitly.

MQPMO_NO_SYNCPOINT must *not* be specified with MQPMO_SYNCPOINT.

This option is not supported on VSE/ESA.

Message-identifier and correlation-identifier options: The following options request the queue manager to generate a new message identifier or correlation identifier:

MQPMO_NEW_MSG_ID

Generate a new message identifier.

This option causes the queue manager to replace the contents of the *MsgId* field in MQMD with a new message identifier. This message identifier is sent with the message, and returned to the application on output from the MQPUT or MQPUT1 call.

This option can also be specified when the message is being put to a distribution list; see the description of the *MsgId* field in the MQPMR structure for details.

Using this option relieves the application of the need to reset the *MsgId* field to MQMI_NONE prior to each MQPUT or MQPUT1 call.

This option is supported in the following environments: AIX, HP-UX, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

MQPMO_NEW_CORREL_ID

Generate a new correlation identifier.

This option causes the queue manager to replace the contents of the *CorrelId* field in MQMD with a new correlation identifier. This correlation identifier is sent with the message, and returned to the application on output from the MQPUT or MQPUT1 call.

This option can also be specified when the message is being put to a distribution list; see the description of the *CorrelId* field in the MQPMR structure for details.

MQPMO_NEW_CORREL_ID is useful in situations where the application requires a unique correlation identifier.

This option is supported in the following environments: AIX, HP-UX, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Group and segment options: The following option relates to the processing of messages in groups and segments of logical messages. These definitions may be of help in understanding the option:

Physical message

This is the smallest unit of information that can be placed on or removed from a queue; it often corresponds to the information specified or retrieved

on a single MQPUT, MQPUT1, or MQGET call. Every physical message has its own message descriptor (MQMD). Generally, physical messages are distinguished by differing values for the message identifier (*MsgId* field in MQMD), although this is not enforced by the queue manager.

Logical message

This is a single unit of application information. In the absence of system constraints, a logical message would be the same as a physical message. But where logical messages are extremely large, system constraints may make it advisable or necessary to split a logical message into two or more physical messages, called *segments*.

A logical message that has been segmented consists of two or more physical messages that have the same nonnull group identifier (*GroupId* field in MQMD), and the same message sequence number (*MsgSeqNumber* field in MQMD). The segments are distinguished by differing values for the segment offset (*Offset* field in MQMD), which gives the offset of the data in the physical message from the start of the data in the logical message. Because each segment is a physical message, the segments in a logical message usually have differing message identifiers.

A logical message that has not been segmented, but for which segmentation has been permitted by the sending application, also has a nonnull group identifier, although in this case there is only one physical message with that group identifier if the logical message does not belong to a message group. Logical messages for which segmentation has been inhibited by the sending application have a null group identifier (MQGI_NONE), unless the logical message belongs to a message group.

Message group

This is a set of one or more logical messages that have the same nonnull group identifier. The logical messages in the group are distinguished by differing values for the message sequence number, which is an integer in the range 1 through *n*, where *n* is the number of logical messages in the group. If one or more of the logical messages is segmented, there will be more than *n* physical messages in the group.

MQPMO_LOGICAL_ORDER

Messages in groups and segments of logical messages will be put in logical order.

This option tells the queue manager how the application will put messages in groups and segments of logical messages. It can be specified only on the MQPUT call; it is *not* valid on the MQPUT1 call.

If MQPMO_LOGICAL_ORDER is specified, it indicates that the application will use successive MQPUT calls to:

- Put the segments in each logical message in the order of increasing segment offset, starting from 0, with no gaps.
- Put all of the segments in one logical message before putting the segments in the next logical message.
- Put the logical messages in each message group in the order of increasing message sequence number, starting from 1, with no gaps.
- Put all of the logical messages in one message group before putting logical messages in the next message group.

The above order is called “logical order”.

MQPMO – Options field

Because the application has told the queue manager how it will put messages in groups and segments of logical messages, the application does not have to maintain and update the group and segment information on each MQPUT call, as the queue manager does this. Specifically, it means that the application does not need to set the *GroupId*, *MsgSeqNumber*, and *Offset* fields in MQMD, as the queue manager sets these to the appropriate values. The application need set only the *MsgFlags* field in MQMD, to indicate when messages belong to groups or are segments of logical messages, and to indicate the last message in a group or last segment of a logical message.

Once a message group or logical message has been started, subsequent MQPUT calls must specify the appropriate MQMF_* flags in *MsgFlags* in MQMD. If the application tries to put a message not in a group when there is an unterminated message group, or put a message which is not a segment when there is an unterminated logical message, the call fails with reason code MQRC_INCOMPLETE_GROUP or MQRC_INCOMPLETE_MSG, as appropriate. However, the queue manager retains the information about the current message group and/or current logical message, and the application can terminate them by sending a message (possibly with no application message data) specifying MQMF_LAST_MSG_IN_GROUP and/or MQMF_LAST_SEGMENT as appropriate, before reissuing the MQPUT call to put the message that is not in the group or not a segment.

Table 61 on page 235 shows the combinations of options and flags that are valid, and the values of the *GroupId*, *MsgSeqNumber*, and *Offset* fields that the queue manager uses in each case. Combinations of options and flags that are not shown in the table are not valid. The columns in the table have the following meanings; “Either” means “Yes” or “No”:

LOG ORD

Indicates whether the MQPMO_LOGICAL_ORDER option is specified on the call.

MIG Indicates whether the MQMF_MSG_IN_GROUP or MQMF_LAST_MSG_IN_GROUP option is specified on the call.

SEG Indicates whether the MQMF_SEGMENT or MQMF_LAST_SEGMENT option is specified on the call.

SEG OK

Indicates whether the MQMF_SEGMENTATION_ALLOWED option is specified on the call.

Cur grp

Indicates whether a current message group exists prior to the call.

Cur log msg

Indicates whether a current logical message exists prior to the call.

Other columns

Show the values that the queue manager uses. “Previous” denotes the value used for the field in the previous message for the queue handle.

Table 61. MQPUT options relating to messages in groups and segments of logical messages

Options you specify				Group and log-msg status prior to call		Values the queue manager uses		
LOG ORD	MIG	SEG	SEG OK	Cur grp	Cur log msg	GroupId	MsgSeqNumber	Offset
Yes	No	No	No	No	No	MQGL_NONE	1	0
Yes	No	No	Yes	No	No	New group id	1	0
Yes	No	Yes	Either	No	No	New group id	1	0
Yes	No	Yes	Either	No	Yes	Previous group id	1	Previous offset + previous segment length
Yes	Yes	Either	Either	No	No	New group id	1	0
Yes	Yes	Either	Either	Yes	No	Previous group id	Previous sequence number + 1	0
Yes	Yes	Yes	Either	Yes	Yes	Previous group id	Previous sequence number	Previous offset + previous segment length
No	No	No	No	Either	Either	MQGL_NONE	1	0
No	No	No	Yes	Either	Either	New group id if MQGL_NONE, else value in field	1	0
No	No	Yes	Either	Either	Either	New group id if MQGL_NONE, else value in field	1	Value in field
No	Yes	No	Either	Either	Either	New group id if MQGL_NONE, else value in field	Value in field	0
No	Yes	Yes	Either	Either	Either	New group id if MQGL_NONE, else value in field	Value in field	Value in field

Notes:

- MQPMO_LOGICAL_ORDER is not valid on the MQPUT1 call.
- For the *MsgId* field, the queue manager generates a new message identifier if MQPMO_NEW_MSG_ID or MQGL_NONE is specified, and uses the value in the field otherwise.
- For the *CorrelId* field, the queue manager generates a new correlation identifier if MQPMO_NEW_CORREL_ID is specified, and uses the value in the field otherwise.

When MQPMO_LOGICAL_ORDER is specified, the queue manager requires that all messages in a group and segments in a logical message be put with the same value in the *Persistence* field in MQMD, that is, all must be persistent, or all must be nonpersistent. If this condition is not satisfied, the MQPUT call fails with reason code MQRC_INCONSISTENT_PERSISTENCE.

The MQPMO_LOGICAL_ORDER option affects units of work as follows:

- If the first physical message in a group or logical message is put within a unit of work, all of the other physical messages in the group or logical message must be put within a unit of work, if the same queue handle is used. However, they need not be put within the *same* unit of work. This allows a message group or logical message consisting of many physical messages to be split across two or more consecutive units of work for the queue handle.
- If the first physical message in a group or logical message is *not* put within a unit of work, none of the other physical messages in the group or logical message can be put within a unit of work, if the same queue handle is used.

If these conditions are not satisfied, the MQPUT call fails with reason code MQRC_INCONSISTENT_UOW.

MQPMO – Options field

When MQPMO_LOGICAL_ORDER is specified, the MQMD supplied on the MQPUT call must not be less than MQMD_VERSION_2. If this condition is not satisfied, the call fails with reason code MQRC_WRONG_MD_VERSION.

If MQPMO_LOGICAL_ORDER is *not* specified, messages in groups and segments of logical messages can be put in any order, and it is not necessary to put complete message groups or complete logical messages. It is the application's responsibility to ensure that the *GroupId*, *MsgSeqNumber*, *Offset*, and *MsgFlags* fields have appropriate values.

This is the technique that can be used to restart a message group or logical message in the middle, after a system failure has occurred. When the system restarts, the application can set the *GroupId*, *MsgSeqNumber*, *Offset*, *MsgFlags*, and *Persistence* fields to the appropriate values, and then issue the MQPUT call with MQPMO_SYNCPOINT or MQPMO_NO_SYNCPOINT set as desired, but *without* specifying MQPMO_LOGICAL_ORDER. If this call is successful, the queue manager retains the group and segment information, and subsequent MQPUT calls using that queue handle can specify MQPMO_LOGICAL_ORDER as normal.

The group and segment information that the queue manager retains for the MQPUT call is separate from the group and segment information that it retains for the MQGET call.

For any given queue handle, the application is free to mix MQPUT calls that specify MQPMO_LOGICAL_ORDER with MQPUT calls that do not, but the following points should be noted:

- If MQPMO_LOGICAL_ORDER is *not* specified, each successful MQPUT call causes the queue manager to set the group and segment information for the queue handle to the values specified by the application; this replaces the existing group and segment information retained by the queue manager for the queue handle.
- If MQPMO_LOGICAL_ORDER is *not* specified, the call does not fail if there is a current message group or logical message; the call may however succeed with an MQCC_WARNING completion code. Table 62 shows the various cases that can arise. In these cases, if the completion code is not MQCC_OK, the reason code is one of the following (as appropriate):
 - MQRC_INCOMPLETE_GROUP
 - MQRC_INCOMPLETE_MSG
 - MQRC_INCONSISTENT_PERSISTENCE
 - MQRC_INCONSISTENT_UOW

Note: The queue manager does not check the group and segment information for the MQPUT1 call.

Table 62. Outcome when MQPUT or MQCLOSE call is not consistent with group and segment information

Current call is	Previous call was MQPUT with MQPMO_LOGICAL_ORDER	Previous call was MQPUT without MQPMO_LOGICAL_ORDER
MQPUT with MQPMO_LOGICAL_ORDER	MQCC_FAILED	MQCC_FAILED

Table 62. Outcome when MQPUT or MQCLOSE call is not consistent with group and segment information (continued)

Current call is	Previous call was MQPUT with MQPMO_LOGICAL_ORDER	Previous call was MQPUT without MQPMO_LOGICAL_ORDER
MQPUT without MQPMO_LOGICAL_ORDER	MQCC_WARNING	MQCC_OK
MQCLOSE with an unterminated group or logical message	MQCC_WARNING	MQCC_OK

Applications that simply want to put messages and segments in logical order are recommended to specify MQPMO_LOGICAL_ORDER, as this is the simplest option to use. This option relieves the application of the need to manage the group and segment information, because the queue manager manages that information. However, specialized applications may need more control than provided by the MQPMO_LOGICAL_ORDER option, and this can be achieved by not specifying that option. If this is done, the application must ensure that the *GroupId*, *MsgSeqNumber*, *Offset*, and *MsgFlags* fields in MQMD are set correctly, prior to each MQPUT or MQPUT1 call.

For example, an application that wants to *forward* physical messages that it receives, without regard for whether those messages are in groups or segments of logical messages, should *not* specify MQPMO_LOGICAL_ORDER. There are two reasons for this:

- If the messages are retrieved and put in order, specifying MQPMO_LOGICAL_ORDER will cause a new group identifier to be assigned to the messages, and this may make it difficult or impossible for the originator of the messages to correlate any reply or report messages that result from the message group.
- In a complex network with multiple paths between sending and receiving queue managers, the physical messages may arrive out of order. By specifying neither MQPMO_LOGICAL_ORDER, nor the corresponding MQGMO_LOGICAL_ORDER on the MQGET call, the forwarding application can retrieve and forward each physical message as soon as it arrives, without having to wait for the next one in logical order to arrive.

Applications that generate report messages for messages in groups or segments of logical messages should also not specify MQPMO_LOGICAL_ORDER when putting the report message.

MQPMO_LOGICAL_ORDER can be specified with any of the other MQPMO_* options.

This option is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Context options: The following options control the processing of message context:

MQPMO_NO_CONTEXT

No context is to be associated with the message.

MQPMO – Options field

Both identity and origin context are set to indicate no context. This means that the context fields in MQMD are set to:

- Blanks for character fields
- Nulls for byte fields
- Zeros for numeric fields

This option is not supported on VSE/ESA.

MQPMO_DEFAULT_CONTEXT

Use default context.

The message is to have default context information associated with it, for both identity and origin. The queue manager sets the context fields in the message descriptor as follows:

Field in MQMD	Value used
<i>UserIdentifier</i>	Determined from the environment if possible; set to blanks otherwise.
<i>AccountingToken</i>	Determined from the environment if possible; set to MQACT_NONE otherwise.
<i>ApplIdentityData</i>	Set to blanks.
<i>PutApplType</i>	Determined from the environment.
<i>PutApplName</i>	Determined from the environment if possible; set to blanks otherwise.
<i>PutDate</i>	Set to date when message is put.
<i>PutTime</i>	Set to time when message is put.
<i>ApplOriginData</i>	Set to blanks.

For more information on message context, see the *WebSphere MQ Application Programming Guide*.

This is the default action if no context options are specified.

This option is not supported on VSE/ESA.

MQPMO_PASS_IDENTITY_CONTEXT

Pass identity context from an input queue handle.

The message is to have context information associated with it. Identity context is taken from the queue handle specified in the *Context* field. Origin context information is generated by the queue manager in the same way that it is for MQPMO_DEFAULT_CONTEXT (see above for values).

For more information on message context, see the *WebSphere MQ Application Programming Guide*.

For the MQPUT call, the queue must have been opened with the MQOO_PASS_IDENTITY_CONTEXT option (or an option that implies it). For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the MQOO_PASS_IDENTITY_CONTEXT option.

This option is not supported in the following environments: VSE/ESA, Windows 3.1, Windows 95, Windows 98.

MQPMO_PASS_ALL_CONTEXT

Pass all context from an input queue handle.

The message is to have context information associated with it. Both identity and origin context are taken from the queue handle specified in the *Context* field. For more information on message context, see the *WebSphere MQ Application Programming Guide*.

For the MQPUT call, the queue must have been opened with the MQOO_PASS_ALL_CONTEXT option (or an option that implies it). For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the MQOO_PASS_ALL_CONTEXT option.

This option is not supported in the following environments: VSE/ESA, Windows 3.1, Windows 95, Windows 98.

MQPMO_SET_IDENTITY_CONTEXT

Set identity context from the application.

The message is to have context information associated with it. The application specifies the identity context in the MQMD structure. Origin context information is generated by the queue manager in the same way that it is for MQPMO_DEFAULT_CONTEXT (see above for values). For more information on message context, see the *WebSphere MQ Application Programming Guide*.

For the MQPUT call, the queue must have been opened with the MQOO_SET_IDENTITY_CONTEXT option (or an option that implies it). For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the MQOO_SET_IDENTITY_CONTEXT option.

This option is not supported on VSE/ESA.

MQPMO_SET_ALL_CONTEXT

Set all context from the application.

The message is to have context information associated with it. The application specifies the identity and origin context in the MQMD structure. For more information on message context, see the *WebSphere MQ Application Programming Guide*.

For the MQPUT call, the queue must have been opened with the MQOO_SET_ALL_CONTEXT option. For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the MQOO_SET_ALL_CONTEXT option.

This option is not supported on VSE/ESA.

Only one of the MQPMO*_CONTEXT context options can be specified. If none of these options is specified, MQPMO_DEFAULT_CONTEXT is assumed.

Other options: The following options control authorization checking, and what happens when the queue manager is quiescing:

MQPMO_ALTERNATE_USER_AUTHORITY

Validate with specified user identifier.

This indicates that the *AlternateUserId* field in the *ObjDesc* parameter of the MQPUT1 call contains a user identifier that is to be used to validate authority to put messages on the queue. The call can succeed only if this *AlternateUserId* is authorized to open the queue with the specified options, regardless of whether the user identifier under which the application is running is authorized to do so. (This does not apply to the context options specified, however, which are always checked against the user identifier under which the application is running.)

This option is valid only with the MQPUT1 call.

This option is not supported on VSE/ESA.

MQPMO – Options field

This option is accepted but ignored on: Windows 3.1, Windows 95, Windows 98.

MQPMO_FAIL_IF QUIESCING

Fail if queue manager is quiescing.

This option forces the MQPUT or MQPUT1 call to fail if the queue manager is in the quiescing state.

On z/OS, this option also forces the MQPUT or MQPUT1 call to fail if the connection (for a CICS or IMS application) is in the quiescing state.

The call returns completion code MQCC_FAILED with reason code MQRC_Q_MGR QUIESCING or MQRC_CONNECTION QUIESCING.

This option is not supported on VSE/ESA.

This option is accepted but ignored on: Windows 3.1, Windows 95, Windows 98.

Default option: If none of the options described above is required, the following option can be used:

MQPMO_NONE

No options specified.

This value can be used to indicate that no other options have been specified; all options assume their default values. MQPMO_NONE is defined to aid program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

This is an input field. The initial value of the *Options* field is MQPMO_NONE.

PutMsgRecFields (MQLONG)

Flags indicating which MQPMR fields are present.

This field contains flags that must be set to indicate which MQPMR fields are present in the put message records provided by the application. *PutMsgRecFields* is used only when the message is being put to a distribution list. The field is ignored if *RecsPresent* is zero, or both *PutMsgRecOffset* and *PutMsgRecPtr* are zero.

For fields that are present, the queue manager uses for each destination the values from the fields in the corresponding put message record. For fields that are absent, the queue manager uses the values from the MQMD structure.

One or more of the following flags can be specified to indicate which fields are present in the put message records:

MQPMRF_MSG_ID

Message-identifier field is present.

MQPMRF_CORREL_ID

Correlation-identifier field is present.

MQPMRF_GROUP_ID

Group-identifier field is present.

MQPMRF_FEEDBACK

Feedback field is present.

MQPMRF_ACCOUNTING_TOKEN

Accounting-token field is present.

If this flag is specified, either MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT must be specified in the *Options* field; if this condition is not satisfied, the call fails with reason code MQRC_PMO_RECORD_FLAGS_ERROR.

If no MQPMR fields are present, the following can be specified:

MQPMRF_NONE

No put-message record fields are present.

If this value is specified, either *RecsPresent* must be zero, or both *PutMsgRecOffset* and *PutMsgRecPtr* must be zero.

MQPMRF_NONE is defined to aid program documentation. It is not intended that this constant be used with any other, but as its value is zero, such use cannot be detected.

If *PutMsgRecFields* contains flags which are not valid, or put message records are provided but *PutMsgRecFields* has the value MQPMRF_NONE, the call fails with reason code MQRC_PMO_RECORD_FLAGS_ERROR.

This is an input field. The initial value of this field is MQPMRF_NONE. This field is ignored if *Version* is less than MQPMO_VERSION_2.

PutMsgRecOffset (MQLONG)

Offset of first put message record from start of MQPMO.

This is the offset in bytes of the first MQPMR put message record from the start of the MQPMO structure. The offset can be positive or negative. *PutMsgRecOffset* is used only when the message is being put to a distribution list. The field is ignored if *RecsPresent* is zero.

When the message is being put to a distribution list, an array of one or more MQPMR put message records can be provided in order to specify certain properties of the message for each destination individually; these properties are:

- message identifier
- correlation identifier
- group identifier
- feedback value
- accounting token

It is not necessary to specify all of these properties, but whatever subset is chosen, the fields must be specified in the correct order. See the description of the MQPMR structure for further details.

Usually, there should be as many put message records as there are object records specified by MQOD when the distribution list is opened; each put message record supplies the message properties for the queue identified by the corresponding object record. Queues in the distribution list which fail to open must still have put message records allocated for them at the appropriate positions in the array, although the message properties are ignored in this case.

It is possible for the number of put message records to differ from the number of object records. If there are fewer put message records than object records, the

MQPMO – PutMsgRecOffset field

message properties for the destinations which do not have put message records are taken from the corresponding fields in the message descriptor MQMD. If there are more put message records than object records, the excess are not used (although it must still be possible to access them). Put message records are optional, but if they are supplied there must be *RecsPresent* of them.

The put message records can be provided in a similar way to the object records in MQOD, either by specifying an offset in *PutMsgRecOffset*, or by specifying an address in *PutMsgRecPtr*; for details of how to do this, see the *ObjectRecOffset* field described in Chapter 12, “MQOD – Object descriptor”, on page 211.

No more than one of *PutMsgRecOffset* and *PutMsgRecPtr* can be used; the call fails with reason code MQRC_PUT_MSG_RECORDS_ERROR if both are nonzero.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQPMO_VERSION_2.

PutMsgRecPtr (MQPTR)

Address of first put message record.

This is the address of the first MQPMR put message record. *PutMsgRecPtr* is used only when the message is being put to a distribution list. The field is ignored if *RecsPresent* is zero.

Either *PutMsgRecPtr* or *PutMsgRecOffset* can be used to specify the put message records, but not both; see the description of the *PutMsgRecOffset* field above for details. If *PutMsgRecPtr* is not used, it must be set to the null pointer or null bytes.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQPMO_VERSION_2.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

RecsPresent (MQLONG)

Number of put message records or response records present.

This is the number of MQPMR put message records or MQRR response records that have been provided by the application. This number can be greater than zero only if the message is being put to a distribution list. Put message records and response records are optional – the application need not provide any records, or it can choose to provide records of only one type. However, if the application provides records of both types, it must provide *RecsPresent* records of each type.

The value of *RecsPresent* need not be the same as the number of destinations in the distribution list. If too many records are provided, the excess are not used; if too few records are provided, default values are used for the message properties for those destinations that do not have put message records (see *PutMsgRecOffset* below).

If *RecsPresent* is less than zero, or is greater than zero but the message is not being put to a distribution list, the call fails with reason code MQRC_RECS_PRESENT_ERROR.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQPMO_VERSION_2.

ResolvedQMgrName (MQCHAR48)

Resolved name of destination queue manager.

This is the name of the destination queue manager after name resolution has been performed by the local queue manager. The name returned is the name of the queue manager that owns the queue identified by *ResolvedQName*, and can be the name of the local queue manager.

If *ResolvedQName* is a shared queue that is owned by the queue-sharing group to which the local queue manager belongs, *ResolvedQMgrName* is the name of the queue-sharing group. If the queue is owned by some other queue-sharing group, *ResolvedQName* can be the name of the queue-sharing group or the name of a queue manager that is a member of the queue-sharing group (the nature of the value returned is determined by the queue definitions that exist at the local queue manager).

A nonblank value is returned only if the object is a single queue; if the object is a distribution list, the value returned is undefined.

This is an output field. The length of this field is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ResolvedQName (MQCHAR48)

Resolved name of destination queue.

This is the name of the destination queue after name resolution has been performed by the local queue manager. The name returned is the name of a queue that exists on the queue manager identified by *ResolvedQMgrName*.

A nonblank value is returned only if the object is a single queue; if the object is a distribution list, the value returned is undefined.

This is an output field. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ResponseRecOffset (MQLONG)

Offset of first response record from start of MQPMO.

This is the offset in bytes of the first MQRR response record from the start of the MQPMO structure. The offset can be positive or negative. *ResponseRecOffset* is used only when the message is being put to a distribution list. The field is ignored if *RecsPresent* is zero.

When the message is being put to a distribution list, an array of one or more MQRR response records can be provided in order to identify the queues to which the message was not sent successfully (*CompCode* field in MQRR), and the reason for each failure (*Reason* field in MQRR). The message may not have been sent either because the queue failed to open, or because the put operation failed. The queue manager sets the response records only when the outcome of the call is

MQPMO – ResponseRecOffset field

mixed (that is, some messages were sent successfully while others failed, or all failed but for differing reasons); reason code MQRC_MULTIPLE_REASONS from the call indicates this case. If the same reason code applies to all queues, that reason is returned in the *Reason* parameter of the MQPUT or MQPUT1 call, and the response records are not set.

Usually, there should be as many response records as there are object records specified by MQOD when the distribution list is opened; when necessary, each response record is set to the completion code and reason code for the put to the queue identified by the corresponding object record. Queues in the distribution list which fail to open must still have response records allocated for them at the appropriate positions in the array, although they are set to the completion code and reason code resulting from the open operation, rather than the put operation.

It is possible for the number of response records to differ from the number of object records. If there are fewer response records than object records, it may not be possible for the application to identify all of the destinations for which the put operation failed, or the reasons for the failures. If there are more response records than object records, the excess are not used (although it must still be possible to access them). Response records are optional, but if they are supplied there must be *RecsPresent* of them.

The response records can be provided in a similar way to the object records in MQOD, either by specifying an offset in *ResponseRecOffset*, or by specifying an address in *ResponseRecPtr*; for details of how to do this, see the *ObjectRecOffset* field described in Chapter 12, “MQOD – Object descriptor”, on page 211. However, no more than one of *ResponseRecOffset* and *ResponseRecPtr* can be used; the call fails with reason code MQRC_RESPONSE_RECORDS_ERROR if both are nonzero.

For the MQPUT1 call, this field must be zero. This is because the response information (if requested) is returned in the response records specified by the object descriptor MQOD.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQPMO_VERSION_2.

ResponseRecPtr (MQPTR)

Address of first response record.

This is the address of the first MQRR response record. *ResponseRecPtr* is used only when the message is being put to a distribution list. The field is ignored if *RecsPresent* is zero.

Either *ResponseRecPtr* or *ResponseRecOffset* can be used to specify the response records, but not both; see the description of the *ResponseRecOffset* field above for details. If *ResponseRecPtr* is not used, it must be set to the null pointer or null bytes.

For the MQPUT1 call, this field must be the null pointer or null bytes. This is because the response information (if requested) is returned in the response records specified by the object descriptor MQOD.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQPMO_VERSION_2.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQPMO_STRUC_ID

Identifier for put-message options structure.

For the C programming language, the constant `MQPMO_STRUC_ID_ARRAY` is also defined; this has the same value as `MQPMO_STRUC_ID`, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is `MQPMO_STRUC_ID`.

Timeout (MQLONG)

Reserved.

This is a reserved field; its value is not significant. The initial value of this field is -1.

UnknownDestCount (MQLONG)

Number of messages sent successfully to remote queues.

This is the number of messages that the current `MQPUT` or `MQPUT1` call has sent successfully to queues in the distribution list that resolve to remote queues. Messages that the queue manager retains temporarily in distribution-list form count as the number of individual destinations that those distribution lists contain. This field is also set when putting a message to a single queue which is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is not set if *Version* is less than `MQPMO_VERSION_2`.

Version (MQLONG)

Structure version number.

The value must be one of the following:

MQPMO_VERSION_1

Version-1 put-message options structure.

This version is supported in all environments.

MQPMO_VERSION_2

Version-2 put-message options structure.

This version is supported in the following environments: AIX, HP-UX, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

MQPMO – Version field

Fields that exist only in the more-recent version of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQPMO_CURRENT_VERSION

Current version of put-message options structure.

This is always an input field. The initial value of this field is MQPMO_VERSION_1.

Initial values and language declarations

Table 63. Initial values of fields in MQPMO

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQPMO_STRUC_ID	'PMOb'
<i>Version</i>	MQPMO_VERSION_1	1
<i>Options</i>	MQPMO_NONE	0
<i>Timeout</i>	None	-1
<i>Context</i>	None	0
<i>KnownDestCount</i>	None	0
<i>UnknownDestCount</i>	None	0
<i>InvalidDestCount</i>	None	0
<i>ResolvedQName</i>	None	Null string or blanks
<i>ResolvedQMgrName</i>	None	Null string or blanks
<i>RecsPresent</i>	None	0
<i>PutMsgRecFields</i>	MQPMRF_NONE	0
<i>PutMsgRecOffset</i>	None	0
<i>ResponseRecOffset</i>	None	0
<i>PutMsgRecPtr</i>	None	Null pointer or null bytes
<i>ResponseRecPtr</i>	None	Null pointer or null bytes
Notes:		
1. The symbol 'b' represents a single blank character.		
2. The value 'Null string or blanks' denotes the null string in C, and blank characters in other programming languages.		
3. In the C programming language, the macro variable MQPMO_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure: MQPMO MyPMO = {MQPMO_DEFAULT};		

C declaration

```
typedef struct tagMQPMO MQPMO;
struct tagMQPMO {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQLONG    Options;          /* Options that control the action of
                                MQPUT and MQPUT1 */
    MQLONG    Timeout;          /* Reserved */
    MQHOBJS  Context;          /* Object handle of input queue */
};
```

MQPMO – Language declarations

```
MQLONG   KnownDestCount;    /* Number of messages sent successfully
                             to local queues */
MQLONG   UnknownDestCount;  /* Number of messages sent successfully
                             to remote queues */
MQLONG   InvalidDestCount;  /* Number of messages that could not be
                             sent */
MQCHAR48  ResolvedQName;     /* Resolved name of destination queue */
MQCHAR48  ResolvedQMgrName;  /* Resolved name of destination queue
                             manager */
MQLONG   RecsPresent;        /* Number of put message records or
                             response records present */
MQLONG   PutMsgRecFields;    /* Flags indicating which MQPMR fields
                             are present */
MQLONG   PutMsgRecOffset;    /* Offset of first put message record
                             from start of MQPMO */
MQLONG   ResponseRecOffset; /* Offset of first response record from
                             start of MQPMO */
MQPTR     PutMsgRecPtr;      /* Address of first put message
                             record */
MQPTR     ResponseRecPtr;    /* Address of first response record */
};
```

COBOL declaration

```
** MQPMO structure
10 MQPMO.
** Structure identifier
15 MQPMO-STRUCID          PIC X(4).
** Structure version number
15 MQPMO-VERSION         PIC S9(9) BINARY.
** Options that control the action of MQPUT and MQPUT1
15 MQPMO-OPTIONS         PIC S9(9) BINARY.
** Reserved
15 MQPMO-TIMEOUT         PIC S9(9) BINARY.
** Object handle of input queue
15 MQPMO-CONTEXT         PIC S9(9) BINARY.
** Number of messages sent successfully to local queues
15 MQPMO-KNOWNDSTCOUNT  PIC S9(9) BINARY.
** Number of messages sent successfully to remote queues
15 MQPMO-UNKNOWNDSTCOUNT PIC S9(9) BINARY.
** Number of messages that could not be sent
15 MQPMO-INVALIDDSTCOUNT PIC S9(9) BINARY.
** Resolved name of destination queue
15 MQPMO-RESOLVEDQNAME   PIC X(48).
** Resolved name of destination queue manager
15 MQPMO-RESOLVEDQMGRNAME PIC X(48).
** Number of put message records or response records present
15 MQPMO-RECSPRESENT     PIC S9(9) BINARY.
** Flags indicating which MQPMR fields are present
15 MQPMO-PUTMSGRECFIELDS PIC S9(9) BINARY.
** Offset of first put message record from start of MQPMO
15 MQPMO-PUTMSGRECOFFSET PIC S9(9) BINARY.
** Offset of first response record from start of MQPMO
15 MQPMO-RESPONSERECOFFSET PIC S9(9) BINARY.
** Address of first put message record
15 MQPMO-PUTMSGRECPTTR   POINTER.
** Address of first response record
15 MQPMO-RESPONSERECPTR  POINTER.
```

PL/I declaration

```
dc1
1 MQPMO based,
3 StrucId          char(4),          /* Structure identifier */
3 Version          fixed bin(31), /* Structure version number */
3 Options          fixed bin(31), /* Options that control the action
                                of MQPUT and MQPUT1 */
```

MQPMO – Language declarations

```
3 Timeout          fixed bin(31), /* Reserved */
3 Context          fixed bin(31), /* Object handle of input queue */
3 KnownDestCount   fixed bin(31), /* Number of messages sent
                                successfully to local queues */
3 UnknownDestCount fixed bin(31), /* Number of messages sent
                                successfully to remote queues */
3 InvalidDestCount fixed bin(31), /* Number of messages that could
                                not be sent */
3 ResolvedQName    char(48),      /* Resolved name of destination
                                queue */
3 ResolvedQMgrName char(48),      /* Resolved name of destination
                                queue manager */
3 RecsPresent      fixed bin(31), /* Number of put message records or
                                response records present */
3 PutMsgRecFields  fixed bin(31), /* Flags indicating which MQPMR
                                fields are present */
3 PutMsgRecOffset  fixed bin(31), /* Offset of first put message
                                record from start of MQPMO */
3 ResponseRecOffset fixed bin(31), /* Offset of first response record
                                from start of MQPMO */
3 PutMsgRecPtr     pointer,        /* Address of first put message
                                record */
3 ResponseRecPtr   pointer;        /* Address of first response
                                record */
```

System/390 assembler declaration

```
MQPMO          DSECT
MQPMO_STRUCID  DS   CL4   Structure identifier
MQPMO_VERSION  DS   F     Structure version number
MQPMO_OPTIONS  DS   F     Options that control the action of
*              MQPUT and MQPUT1
MQPMO_TIMEOUT  DS   F     Reserved
MQPMO_CONTEXT  DS   F     Object handle of input queue
MQPMO_KNOWNDSTCOUNT DS F   Number of messages sent successfully
*              to local queues
MQPMO_UNKNOWNDSTCOUNT DS F   Number of messages sent successfully
*              to remote queues
MQPMO_INVALIDDESTCOUNT DS F   Number of messages that could not be
*              sent
MQPMO_RESOLVEDQNAME DS   CL48 Resolved name of destination queue
MQPMO_RESOLVEDQMGRNAME DS CL48 Resolved name of destination queue
*              manager
*
MQPMO_LENGTH  EQU   *-MQPMO
              ORG   MQPMO
MQPMO_AREA    DS   CL(MQPMO_LENGTH)
```

TAL declaration

```
STRUCT        MQPMO^DEF (*);
BEGIN
STRUCT        STRUCID;
BEGIN STRING  BYTE [0:3]; END;
INT(32)       VERSION;
INT(32)       OPTIONS;
INT(32)       TIMEOUT;
INT(32)       CONTEXT;
INT(32)       KNOWNDSTCOUNT;
INT(32)       UNKNOWNDSTCOUNT;
INT(32)       INVALIDDESTCOUNT;
STRUCT        RESOLVEDQNAME;
BEGIN STRING  BYTE [0:47]; END;
STRUCT        RESOLVEDQMGRNAME;
BEGIN STRING  BYTE [0:47]; END;
END;
```

Visual Basic declaration

```

Type MQPMO
  StrucId      As String*4  'Structure identifier'
  Version     As Long      'Structure version number'
  Options     As Long      'Options that control the action of'
                    'MQPUT and MQPUT1'
  Timeout     As Long      'Reserved'
  Context     As Long      'Object handle of input queue'
  KnownDestCount As Long   'Number of messages sent successfully'
                    'to local queues'
  UnknownDestCount As Long 'Number of messages sent successfully'
                    'to remote queues'
  InvalidDestCount As Long 'Number of messages that could not be'
                    'sent'
  ResolvedQName As String*48 'Resolved name of destination queue'
  ResolvedQMgrName As String*48 'Resolved name of destination queue'
                    'manager'
  RecsPresent As Long      'Number of put message records or'
                    'response records present'
  PutMsgRecFields As Long   'Flags indicating which MQPMR fields'
                    'are present'
  PutMsgRecOffset As Long   'Offset of first put message record'
                    'from start of MQPMO'
  ResponseRecOffset As Long 'Offset of first response record from'
                    'start of MQPMO'
  PutMsgRecPtr As MQPTR    'Address of first put message record'
  ResponseRecPtr As MQPTR  'Address of first response record'
End Type

```

MQPMO – Language declarations

Chapter 15. MQPMR – Put-message record

The following table summarizes the fields in the structure.

Table 64. Fields in MQPMR

Field	Description	Page
<i>MsgId</i>	Message identifier	253
<i>CorrelId</i>	Correlation identifier	252
<i>GroupId</i>	Group identifier	252
<i>Feedback</i>	Feedback or reason code	252
<i>AccountingToken</i>	Accounting token	252

Overview

Availability: AIX, HP-UX, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Purpose: The MQPMR structure is used to specify various message properties for a single destination when a message is being put to a distribution list. MQPMR is an input/output structure for the MQPUT and MQPUT1 calls.

Character set and encoding: Data in MQPMR must be in the character set and encoding of the local queue manager; these are given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE, respectively. However, if the application is running as an MQ client, the structure must be in the character set and encoding of the client.

Usage: By providing an array of these structures on the MQPUT or MQPUT1 call, it is possible to specify different values for each destination queue in a distribution list. Some of the fields are input only, others are input/output.

Note: This structure is unusual in that it does not have a fixed layout. The fields in this structure are optional, and the presence or absence of each field is indicated by the flags in the *PutMsgRecFields* field in MQPMO. Fields that are present *must occur in the following order*:

MsgId
CorrelId
GroupId
Feedback
AccountingToken

Fields that are absent occupy no space in the record.

Because MQPMR does not have a fixed layout, no definition of it is provided in the header, COPY, and INCLUDE files for the supported programming languages. The application programmer should create a declaration containing the fields that are required by the application, and set the flags in *PutMsgRecFields* to indicate the fields that are present.

Fields

The MQPMR structure contains the following fields; the fields are described in **alphabetic order**:

AccountingToken (MQBYTE32)

Accounting token.

This is the accounting token to be used for the message sent to the queue whose name was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *AccountingToken* field in MQMD for a put to a single queue. See the description of *AccountingToken* in Chapter 10, “MQMD – Message descriptor”, on page 141 for information about the content of this field.

If this field is not present, the value in MQMD is used.

This is an input field.

CorrelId (MQBYTE24)

Correlation identifier.

This is the correlation identifier to be used for the message sent to the queue whose name was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *CorrelId* field in MQMD for a put to a single queue.

If this field is not present in the MQPMR record, or there are fewer MQPMR records than destinations, the value in MQMD is used for those destinations that do not have an MQPMR record containing a *CorrelId* field.

If MQPMO_NEW_CORREL_ID is specified, a *single* new correlation identifier is generated and used for all of the destinations in the distribution list, regardless of whether they have MQPMR records. This is different from the way that MQPMO_NEW_MSG_ID is processed (see *MsgId* field).

This is an input/output field.

Feedback (MQLONG)

Feedback or reason code.

This is the feedback code to be used for the message sent to the queue whose name was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *Feedback* field in MQMD for a put to a single queue.

If this field is not present, the value in MQMD is used.

This is an input field.

GroupId (MQBYTE24)

Group identifier.

This is the group identifier to be used for the message sent to the queue whose name was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *GroupId* field in MQMD for a put to a single queue.

If this field is not present in the MQPMR record, or there are fewer MQPMR records than destinations, the value in MQMD is used for those destinations that do not have an MQPMR record containing a *GroupId* field. The value is processed as documented in Table 61 on page 235, but with the following differences:

- In those cases where a new group identifier would be used, the queue manager generates a different group identifier for each destination (that is, no two destinations have the same group identifier).
- In those cases where the value in the field would be used, the call fails with reason code MQRC_GROUP_ID_ERROR.

This is an input/output field.

MsgId (MQBYTE24)

Message identifier.

This is the message identifier to be used for the message sent to the queue whose name was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *MsgId* field in MQMD for a put to a single queue.

If this field is not present in the MQPMR record, or there are fewer MQPMR records than destinations, the value in MQMD is used for those destinations that do not have an MQPMR record containing a *MsgId* field. If that value is MQMI_NONE, a new message identifier is generated for *each* of those destinations (that is, no two of those destinations have the same message identifier).

If MQPMO_NEW_MSG_ID is specified, new message identifiers are generated for all of the destinations in the distribution list, regardless of whether they have MQPMR records. This is different from the way that MQPMO_NEW_CORREL_ID is processed (see *CorrelId* field).

This is an input/output field.

Initial values and language declarations

There are no initial values defined for this structure, as no structure declarations are provided in the header, COPY, and INCLUDE files for the supported programming languages. The sample declarations below show how the structure should be declared by the application programmer if all of the fields are required.

C declaration

```
typedef struct tagMQPMR MQPMR;
struct tagMQPMR {
    MQBYTE24  MsgId;           /* Message identifier */
    MQBYTE24  CorrelId;       /* Correlation identifier */
    MQBYTE24  GroupId;        /* Group identifier */
    MQLONG    Feedback;       /* Feedback or reason code */
    MQBYTE32  AccountingToken; /* Accounting token */
};
```

MQPMR – Language declarations

COBOL declaration

```
** MQPMR structure
10 MQPMR.
** Message identifier
15 MQPMR-MSGID PIC X(24).
** Correlation identifier
15 MQPMR-CORRELID PIC X(24).
** Group identifier
15 MQPMR-GROUPID PIC X(24).
** Feedback or reason code
15 MQPMR-FEEDBACK PIC S9(9) BINARY.
** Accounting token
15 MQPMR-ACCOUNTINGTOKEN PIC X(32).
```

PL/I declaration

```
dc1
1 MQPMR based,
3 MsgId char(24), /* Message identifier */
3 CorrelId char(24), /* Correlation identifier */
3 GroupId char(24), /* Group identifier */
3 Feedback fixed bin(31), /* Feedback or reason code */
3 AccountingToken char(32); /* Accounting token */
```

Visual Basic declaration

```
Type MQPMR
MsgId As MQBYTE24 'Message identifier'
CorrelId As MQBYTE24 'Correlation identifier'
GroupId As MQBYTE24 'Group identifier'
Feedback As Long 'Feedback or reason code'
AccountingToken As MQBYTE32 'Accounting token'
End Type
```

Chapter 16. MQRFH – Rules and formatting header

The following table summarizes the fields in the structure.

Table 65. Fields in MQRFH

Field	Description	Page
<i>StrucId</i>	Structure identifier	257
<i>Version</i>	Structure version number	258
<i>StrucLength</i>	Total length of MQRFH including string containing name/value pairs	258
<i>Encoding</i>	Numeric encoding of data that follows <i>NameValueString</i>	256
<i>CodedCharSetId</i>	Character set identifier of data that follows <i>NameValueString</i>	255
<i>Format</i>	Format name of data that follows <i>NameValueString</i>	256
<i>Flags</i>	Flags	256
<i>NameValueString</i>	String containing name/value pairs	257

Overview

Availability: AIX, HP-UX, z/OS, OS/2, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Purpose: The MQRFH structure defines the layout of the rules and formatting header. This header can be used to send string data in the form of name/value pairs.

Format name: MQFMT_RF_HEADER.

Character set and encoding: The fields in the MQRFH structure (including *NameValueString*) are in the character set and encoding given by the *CodedCharSetId* and *Encoding* fields in the header structure that precedes the MQRFH, or by those fields in the MQMD structure if the MQRFH is at the start of the application message data.

The character set must be one that has single-byte characters for the characters that are valid in queue names.

Fields

The MQRFH structure contains the following fields; the fields are described in **alphabetic order**:

CodedCharSetId (MQLONG)

Character set identifier of data that follows *NameValueString*.

This specifies the character set identifier of the data that follows *NameValueString*; it does not apply to character data in the MQRFH structure itself.

MQRFH – CodedCharSetId field

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

MQCCSI_INHERIT

Inherit character-set identifier of this structure.

Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

MQCCSI_INHERIT cannot be used if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

This value is supported in the following environments: AIX, HP-UX, z/OS, OS/2, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG)

Numeric encoding of data that follows *NameValueString*.

This specifies the numeric encoding of the data that follows *NameValueString*; it does not apply to numeric data in the MQRFH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is MQENC_NATIVE.

Flags (MQLONG)

Flags.

The following can be specified:

MQRFH_NONE

No flags.

The initial value of this field is MQRFH_NONE.

Format (MQCHAR8)

Format name of data that follows *NameValueString*.

This specifies the format name of the data that follows *NameValueString*.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *Format* field in MQMD.

The initial value of this field is MQFMT_NONE.

NameValueString (MQCHARn)

String containing name/value pairs.

This is a variable-length character string containing name/value pairs in the form:
name1 value1 name2 value2 name3 value3 ...

Each name or value must be separated from the adjacent name or value by one or more blank characters; these blanks are not significant. A name or value can contain significant blanks by prefixing and suffixing the name or value with the double-quote character; all characters between the open double-quote and the matching close double-quote are treated as significant. In the following example, the name is FAMOUS_WORDS, and the value is Hello World:

```
FAMOUS_WORDS "Hello World"
```

A name or value can contain any characters other than the null character (which acts as a delimiter for *NameValueString* – see below). However, to assist interoperability an application may prefer to restrict names to the following characters:

- First character: upper or lowercase alphabetic (A through Z, or a through z), or underscore.
- Subsequent characters: upper or lowercase alphabetic, decimal digit (0 through 9), underscore, hyphen, or dot.

If a name or value contains one or more double-quote characters, the name or value must be enclosed in double quotes, and each double quote within the string must be doubled:

```
Famous_Words "The program displayed ""Hello World"""
```

Names and values are case sensitive, that is, lowercase letters are not considered to be the same as uppercase letters. For example, FAMOUS_WORDS and Famous_Words are two different names.

The length in bytes of *NameValueString* is equal to *StrucLength* minus MQRFH_STRUC_LENGTH_FIXED. To avoid problems with data conversion of the user data in some environments, it is recommended that this length should be a multiple of four. *NameValueString* must be padded with blanks to this length, or terminated earlier by placing a null character following the last significant character in the string. The null character and the bytes following it, up to the specified length of *NameValueString*, are ignored.

Note: Because the length of this field is not fixed, the field is omitted from the declarations of the structure that are provided for the supported programming languages.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQRFH_STRUC_ID

Identifier for rules and formatting header structure.

For the C programming language, the constant MQRFH_STRUC_ID_ARRAY is also defined; this has the same value as MQRFH_STRUC_ID, but is an array of characters instead of a string.

MQRFH – StrucLength field

The initial value of this field is `MQRFH_STRUC_ID`.

StrucLength (MQLONG)

Total length of MQRFH including *NameValueString*.

This is the length in bytes of the MQRFH structure, including the *NameValueString* field at the end of the structure. The length does *not* include any user data that follows the *NameValueString* field.

To avoid problems with data conversion of the user data in some environments, it is recommended that *StrucLength* should be a multiple of four.

The following constant gives the length of the *fixed* part of the structure, that is, the length excluding the *NameValueString* field:

`MQRFH_STRUC_LENGTH_FIXED`

Length of fixed part of MQRFH structure.

The initial value of this field is `MQRFH_STRUC_LENGTH_FIXED`.

Version (MQLONG)

Structure version number.

The value must be:

`MQRFH_VERSION_1`

Version-1 rules and formatting header structure.

The initial value of this field is `MQRFH_VERSION_1`.

Initial values and language declarations

Table 66. Initial values of fields in MQRFH

Field name	Name of constant	Value of constant
<i>StrucId</i>	<code>MQRFH_STRUC_ID</code>	'RFHb'
<i>Version</i>	<code>MQRFH_VERSION_1</code>	1
<i>StrucLength</i>	<code>MQRFH_STRUC_LENGTH_FIXED</code>	32
<i>Encoding</i>	<code>MQENC_NATIVE</code>	Depends on environment
<i>CodedCharSetId</i>	<code>MQCCSI_UNDEFINED</code>	0
<i>Format</i>	<code>MQFMT_NONE</code>	Blanks
<i>Flags</i>	<code>MQRFH_NONE</code>	0
Notes: <ol style="list-style-type: none">1. The symbol 'b' represents a single blank character.2. In the C programming language, the macro variable <code>MQRFH_DEFAULT</code> contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure: <pre>MQRFH MyRFH = {MQRFH_DEFAULT};</pre>		

C declaration

```
typedef struct tagMQRFH MQRFH;
struct tagMQRFH {
    MQCHAR4  StrucId;           /* Structure identifier */
    MQLONG   Version;          /* Structure version number */
    MQLONG   StrucLength;      /* Total length of MQRFH including
                               NameValueString */
    MQLONG   Encoding;         /* Numeric encoding of data that follows
                               NameValueString */
    MQLONG   CodedCharSetId;   /* Character set identifier of data that
                               follows NameValueString */
    MQCHAR8  Format;           /* Format name of data that follows
                               NameValueString */
    MQLONG   Flags;           /* Flags */
};
```

COBOL declaration

```
** MQRFH structure
10 MQRFH.
** Structure identifier
15 MQRFH-STRUCID PIC X(4).
** Structure version number
15 MQRFH-VERSION PIC S9(9) BINARY.
** Total length of MQRFH including NAMEVALUESTRING
15 MQRFH-STRUCLength PIC S9(9) BINARY.
** Numeric encoding of data that follows NAMEVALUESTRING
15 MQRFH-ENCODING PIC S9(9) BINARY.
** Character set identifier of data that follows NAMEVALUESTRING
15 MQRFH-CODEDCHARSETID PIC S9(9) BINARY.
** Format name of data that follows NAMEVALUESTRING
15 MQRFH-FORMAT PIC X(8).
** Flags
15 MQRFH-FLAGS PIC S9(9) BINARY.
```

PL/I declaration

```
dcl
1 MQRFH based,
3 StrucId char(4), /* Structure identifier */
3 Version fixed bin(31), /* Structure version number */
3 StrucLength fixed bin(31), /* Total length of MQRFH including
                               NameValueString */
3 Encoding fixed bin(31), /* Numeric encoding of data that
                               follows NameValueString */
3 CodedCharSetId fixed bin(31), /* Character set identifier of data that
                               follows NameValueString */
3 Format char(8), /* Format name of data that follows
                               NameValueString */
3 Flags fixed bin(31); /* Flags */
```

System/390 assembler declaration

```
MQRFH DSECT
MQRFH_STRUCID DS CL4 Structure identifier
MQRFH_VERSION DS F Structure version number
MQRFH_STRUCLength DS F Total length of MQRFH including
* NAMEVALUESTRING
MQRFH_ENCODING DS F Numeric encoding of data that follows
* NAMEVALUESTRING
MQRFH_CODEDCHARSETID DS F Character set identifier of data that
* follows NAMEVALUESTRING
MQRFH_FORMAT DS CL8 Format name of data that follows
* NAMEVALUESTRING
MQRFH_FLAGS DS F Flags
*
```

MQRFH – Language declarations

```
MQRFH_LENGTH      EQU  *-MQRFH
                   ORG  MQRFH
MQRFH_AREA        DS   CL(MQRFH_LENGTH)
```

Visual Basic declaration

```
Type MQRFH
  StrucId      As String*4 'Structure identifier'
  Version     As Long      'Structure version number'
  StrucLength  As Long      'Total length of MQRFH including'
                                     'NameValueString'
  Encoding    As Long      'Numeric encoding of data that follows'
                                     'NameValueString'
  CodedCharSetId As Long    'Character set identifier of data that'
                                     'follows NameValueString'
  Format       As String*8  'Format name of data that follows'
                                     'NameValueString'
  Flags       As Long      'Flags'
End Type
```

Chapter 17. MQRFH2 – Rules and formatting header 2

The following table summarizes the fields in the structure.

Table 67. Fields in MQRFH2

Field	Description	Page
<i>StrucId</i>	Structure identifier	266
<i>Version</i>	Structure version number	267
<i>StrucLength</i>	Total length of MQRFH2 including all <i>NameValueLength</i> and <i>NameValueData</i> fields	266
<i>Encoding</i>	Numeric encoding of data that follows <i>NameValueData</i>	262
<i>CodedCharSetId</i>	Character set identifier of data that follows <i>NameValueData</i>	262
<i>Format</i>	Format name of data that follows <i>NameValueData</i>	263
<i>Flags</i>	Flags	263
<i>NameValueCCSID</i>	Character set identifier of <i>NameValueData</i>	263
<i>NameValueLength</i>	Length of <i>NameValueData</i>	266
<i>NameValueData</i>	Name/value data	263

Overview

Availability: AIX, HP-UX, z/OS, OS/2, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Purpose: The MQRFH2 header is based on the MQRFH header, but it allows Unicode strings to be transported without translation, and it can carry numeric datatypes.

The MQRFH2 structure defines the format of the version-2 rules and formatting header. This header can be used to send data that has been encoded using an XML-like syntax. A message can contain two or more MQRFH2 structures in series, with user data optionally following the last MQRFH2 structure in the series.

Format name: MQFMT_RF_HEADER_2.

Character set and encoding: Special rules apply to the character set and encoding used for the MQRFH2 structure:

- Fields other than *NameValueData* are in the character set and encoding given by the *CodedCharSetId* and *Encoding* fields in the header structure that precedes MQRFH2, or by those fields in the MQMD structure if the MQRFH2 is at the start of the application message data.

The character set must be one that has single-byte characters for the characters that are valid in queue names.

When MQGMO_CONVERT is specified on the MQGET call, the queue manager converts these fields to the requested character set and encoding.

MQRFH2 – Rules and formatting header 2

- *NameValueData* is in the character set given by the *NameValueCCSID* field. Only certain Unicode character sets are valid for *NameValueCCSID* (see the description of *NameValueCCSID* for details).

Some character sets have a representation that is dependent on the encoding. If *NameValueCCSID* is one of these character sets, *NameValueData* must be in the same encoding as the other fields in the MQRFH2.

When MQGMO_CONVERT is specified on the MQGET call, the queue manager converts *NameValueData* to the requested encoding, but does not change its character set.

Fields

The MQRFH2 structure contains the following fields; the fields are described in **alphabetic order**:

CodedCharSetId (MQLONG)

Character set identifier of data that follows last *NameValueData* field.

This specifies the character set identifier of the data that follows the last *NameValueData* field; it does not apply to character data in the MQRFH2 structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

MQCCSI_INHERIT

Inherit character-set identifier of this structure.

Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

MQCCSI_INHERIT cannot be used if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

This value is supported in the following environments: AIX, HP-UX, z/OS, OS/2, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

The initial value of this field is MQCCSI_INHERIT.

Encoding (MQLONG)

Numeric encoding of data that follows last *NameValueData* field.

This specifies the numeric encoding of the data that follows the last *NameValueData* field; it does not apply to numeric data in the MQRFH2 structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is MQENC_NATIVE.

Flags (MQLONG)

Flags.

The following value must be specified:

MQRFH_NONE
No flags.

The initial value of this field is MQRFH_NONE.

Format (MQCHAR8)

Format name of data that follows last *NameValueData* field.

This specifies the format name of the data that follows the last *NameValueData* field.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *Format* field in MQMD.

The initial value of this field is MQFMT_NONE.

NameValueCCSID (MQLONG)

Character set identifier of *NameValueData*.

This specifies the coded character set identifier of the data in the *NameValueData* field. This is different from the character set of the other strings in the MQRFH2 structure, and can be different from the character set of the data (if any) that follows the last *NameValueData* field at the end of the structure.

NameValueCCSID must have one of the following values:

CCSID	Meaning
1200	UCS-2 open-ended
13488	UCS-2 2.0 subset
17584	UCS-2 2.1 subset (includes the Euro symbol)
1208	UTF-8

For the UCS-2 character sets, the encoding (byte order) of the *NameValueData* must be the same as the encoding of the other fields in the MQRFH2 structure. Surrogate characters (X'D800' through X'DFFF') are not supported.

Note: If *NameValueCCSID* does not have one of the values listed above, and the MQRFH2 structure requires conversion on the MQGET call, the call completes with reason code MQRC_SOURCE_CCSID_ERROR and the message is returned unconverted.

The initial value of this field is 1208.

NameValueData (MQCHARn)

Name/value data.

MQRFH2 – NameValueData field

This is a variable-length character string containing data encoded using an XML-like syntax. The length in bytes of this string is given by the *NameValueLength* field that precedes the *NameValueData* field; this length should be a multiple of four.

The *NameValueLength* and *NameValueData* fields are optional, but if present they must occur as a pair and be adjacent. The pair of fields can be repeated as many times as required, for example:

```
length1 data1 length2 data2 length3 data3
```

Because these fields are optional, they are omitted from the declarations of the structure that are provided for the various programming languages supported.

NameValueData is unusual because it is *not* converted to the character set specified on the MQGET call when the message is retrieved with the MQGMO_CONVERT option in effect; *NameValueData* remains in its original character set. However, *NameValueData* is converted to the encoding specified on the MQGET call.

Syntax of name/value data: The string consists of a single “folder” that contains zero or more properties. The folder is delimited by XML start and end tags whose name is the name of the folder:

```
<folder> property1 property2 ... </folder>
```

Characters following the folder end tag, up to the length defined by *NameValueLength*, must be blank. Within the folder, each property is composed of a name and a value, and optionally a data type:

```
<name dt="datatype">value</name>
```

In these examples:

- The delimiter characters (<, =, ", /, and >) must be specified exactly as shown.
- name is the user-specified name of the property; see below for more information about names.
- datatype is an optional user-specified data type of the property; see below for valid data types.
- value is the user-specified value of the property; see below for more information about values.
- Blanks are significant between the > character which precedes a value, and the < character which follows the value, and at least one blank must precede dt=. Elsewhere blanks can be coded freely between tags, or preceding or following tags (for example, in order to improve readability); these blanks are not significant.

If properties are related to each other, they can be grouped together by enclosing them within XML start and end tags whose name is the name of the group:

```
<folder> <group> property1 property2 ... </group> </folder>
```

Groups can be nested within other groups, without limit, and a given group can occur more than once within a folder. It is also valid for a folder to contain some properties in groups and other properties not in groups.

Names of properties, groups, and folders: Names of properties, groups, and folders must be valid XML tag names, with the exception of the colon character, which is not permitted in a property, group, or folder name. In particular:

MQRFH2 – NameValueData field

- Names must start with a letter or an underscore. Valid letters are defined in the W3C XML specification, and consist essentially of Unicode categories Ll, Lu, Lo, Lt, and Nl.
- The remaining characters in a name can be letters, decimal digits, underscores, hyphens, or dots. These correspond to Unicode categories Ll, Lu, Lo, Lt, Nl, Mc, Mn, Lm, and Nd.
- The Unicode compatibility characters (X'F900' and above) are not permitted in any part of a name.
- Names must not start with the string XML in any mixture of upper or lowercase.

In addition:

- Names are case-sensitive. For example, ABC, abc, and Abc are three different names.
- Each folder has a separate name space. As a result, a group or property in one folder does not conflict with a group or property of the same name in another folder.
- Groups and properties occupy the same name space within a folder. As a result, a property cannot have the same name as a group within the folder containing that property.

Generally, programs that analyze the *NameValueData* field should ignore properties or groups that have names that the program does not recognize, provided that those properties or groups are correctly formed.

Data types of properties: Each property can have an optional data type. If specified, the data type must be one of the following values, in upper, lower, or mixed case:

Data type	Used for
string	Any sequence of characters. Certain characters must be specified using escape sequences (see below).
boolean	The character 0 or 1 (1 denotes TRUE).
bin.hex	Hexadecimal digits representing octets.
i1	Integer number in the range -128 through +127, expressed using only decimal digits and optional sign.
i2	Integer number in the range -32 768 through +32 767, expressed using only decimal digits and optional sign.
i4	Integer number in the range -2 147 483 648 through +2 147 483 647, expressed using only decimal digits and optional sign.
i8	Integer number in the range -9 223 372 036 854 775 808 through +9 223 372 036 854 775 807, expressed using only decimal digits and optional sign.
int	Integer number in the range -9 223 372 036 854 775 808 through +9 223 372 036 854 775 807, expressed using only decimal digits and optional sign. This can be used in place of i1, i2, i4, or i8 if the sender does not wish to imply a particular precision.
r4	Floating-point number with magnitude in the range 1.175E-37 through 3.402 823 47E+38, expressed using decimal digits, optional sign, optional fractional digits, and optional exponent.
r8	Floating-point number with magnitude in the range 2.225E-307 through 1.797 693 134 862 3E+308 expressed using decimal digits, optional sign, optional fractional digits, and optional exponent.

Values of properties: The value of a property can consist of any characters, except as detailed below. Each occurrence in the value of a character marked as

MQRFH2 – NameValueData field

“mandatory” must be replaced by the corresponding escape sequence. Each occurrence in the value of a character marked as “optional” can be replaced by the corresponding escape sequence, but this is not required.

Character	Escape sequence	Usage
&	&	Mandatory
<	<	Mandatory
>	>	Optional
"	"	Optional
'	'	Optional

Note: The & character at the start of an escape sequence must *not* be replaced by &.

In the following example, the blanks in the value are significant; however, no escape sequences are needed:

```
<Famous_Words>The program displayed "Hello World"</Famous_Words>
```

NameValueLength (MQLONG)

Length of *NameValueData*.

This specifies the length in bytes of the data in the *NameValueData* field. To avoid problems with data conversion of the data (if any) that *follows* the *NameValueData* field, *NameValueLength* should be a multiple of four.

Note: The *NameValueLength* and *NameValueData* fields are optional, but if present they must occur as a pair and be adjacent. The pair of fields can be repeated as many times as required, for example:

```
length1 data1 length2 data2 length3 data3
```

Because these fields are optional, they are omitted from the declarations of the structure that are provided for the various programming languages supported.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQRFH_STRUC_ID

Identifier for rules and formatting header structure.

For the C programming language, the constant `MQRFH_STRUC_ID_ARRAY` is also defined; this has the same value as `MQRFH_STRUC_ID`, but is an array of characters instead of a string.

The initial value of this field is `MQRFH_STRUC_ID`.

StrucLength (MQLONG)

Total length of MQRFH2 including all *NameValueLength* and *NameValueData* fields.

This is the length in bytes of the MQRFH2 structure, including the *NameValueLength* and *NameValueData* fields at the end of the structure. It is valid for there to be multiple pairs of *NameValueLength* and *NameValueData* fields at the end of the structure, in the sequence:

MQRFH2 – StrucLength field

length1, data1, length2, data2, ...

StrucLength does *not* include any user data that may follow the last *NameValueData* field at the end of the structure.

To avoid problems with data conversion of the user data in some environments, it is recommended that *StrucLength* should be a multiple of four.

The following constant gives the length of the *fixed* part of the structure, that is, the length excluding the *NameValueLength* and *NameValueData* fields:

MQRFH_STRUC_LENGTH_FIXED_2

Length of fixed part of MQRFH2 structure.

The initial value of this field is `MQRFH_STRUC_LENGTH_FIXED_2`.

Version (MQLONG)

Structure version number.

The value must be:

MQRFH_VERSION_2

Version-2 rules and formatting header structure.

The initial value of this field is `MQRFH_VERSION_2`.

Initial values and language declarations

Table 68. Initial values of fields in MQRFH2

Field name	Name of constant	Value of constant
<i>StrucId</i>	<code>MQRFH_STRUC_ID</code>	'RFHb'
<i>Version</i>	<code>MQRFH_VERSION_2</code>	2
<i>StrucLength</i>	<code>MQRFH_STRUC_LENGTH_FIXED_2</code>	36
<i>Encoding</i>	<code>MQENC_NATIVE</code>	Depends on environment
<i>CodedCharSetId</i>	<code>MQCCSI_INHERIT</code>	-2
<i>Format</i>	<code>MQFMT_NONE</code>	Blanks
<i>Flags</i>	<code>MQRFH_NONE</code>	0
<i>NameValueCCSID</i>	None	1208
Notes: <ol style="list-style-type: none">1. The symbol 'b' represents a single blank character.2. In the C programming language, the macro variable <code>MQRFH2_DEFAULT</code> contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure: <pre>MQRFH2 MyRFH2 = {MQRFH2_DEFAULT};</pre>		

C declaration

```
typedef struct tagMQRFH2 MQRFH2;  
struct tagMQRFH2 {  
    MQCHAR4 StrucId;          /* Structure identifier */
```

MQRFH2 – Language declarations

```

MQLONG  Version;          /* Structure version number */
MQLONG  StrucLength;     /* Total length of MQRFH2 including all
                          NameValueLength and NameValueData
                          fields */
MQLONG  Encoding;       /* Numeric encoding of data that follows
                          last NameValueData field */
MQLONG  CodedCharSetId; /* Character set identifier of data that
                          follows last NameValueData field */
MQCHAR8  Format;         /* Format name of data that follows last
                          NameValueData field */
MQLONG  Flags;          /* Flags */
MQLONG  NameValueCCSID; /* Character set identifier of
                          NameValueData */
};
```

COBOL declaration

```

** MQRFH2 structure
10 MQRFH2.
** Structure identifier
15 MQRFH2-STRUCID PIC X(4).
** Structure version number
15 MQRFH2-VERSION PIC S9(9) BINARY.
** Total length of MQRFH2 including all NAMEVALUELENGTH and
** NAMEVALUEDATA fields
15 MQRFH2-STRUCLength PIC S9(9) BINARY.
** Numeric encoding of data that follows last NAMEVALUEDATA field
15 MQRFH2-ENCODING PIC S9(9) BINARY.
** Character set identifier of data that follows last NAMEVALUEDATA
** field
15 MQRFH2-CODEDCHARSETID PIC S9(9) BINARY.
** Format name of data that follows last NAMEVALUEDATA field
15 MQRFH2-FORMAT PIC X(8).
** Flags
15 MQRFH2-FLAGS PIC S9(9) BINARY.
** Character set identifier of NAMEVALUEDATA
15 MQRFH2-NAMEVALUECCSID PIC S9(9) BINARY.
```

PL/I declaration

```

dcl
1 MQRFH2 based,
3 StrucId char(4), /* Structure identifier */
3 Version fixed bin(31), /* Structure version number */
3 StrucLength fixed bin(31), /* Total length of MQRFH2 including
                             all NameValueLength and
                             NameValueData fields */
3 Encoding fixed bin(31), /* Numeric encoding of data that
                             follows last NameValueData field */
3 CodedCharSetId fixed bin(31), /* Character set identifier of data
                             that follows last NameValueData
                             field */
3 Format char(8), /* Format name of data that follows
                  last NameValueData field */
3 Flags fixed bin(31), /* Flags */
3 NameValueCCSID fixed bin(31); /* Character set identifier of
                             NameValueData */
```

System/390 assembler declaration

```

MQRFH          DSECT
MQRFH_STRUCID  DS CL4 Structure identifier
MQRFH_VERSION  DS F   Structure version number
MQRFH_STRUCLNGTH DS F   Total length of MQRFH2 including all
*              NAMEVALUELENGTH and NAMEVALUEDATA fields
MQRFH_ENCODING DS F   Numeric encoding of data that follows
*              last NAMEVALUEDATA field
```

MQRFH2 – Language declarations

```
MQRFH_CODEDCHARSETID DS F Character set identifier of data that
* follows last NAMEVALUEDATA field
MQRFH_FORMAT DS CL8 Format name of data that follows last
* NAMEVALUEDATA field
MQRFH_FLAGS DS F Flags
MQRFH_NAMEVALUECCSID DS F Character set identifier of
* NAMEVALUEDATA
*
MQRFH_LENGTH EQU *-MQRFH
ORG MQRFH
MQRFH_AREA DS CL(MQRFH_LENGTH)
```

Visual Basic declaration

```
Type MQRFH2
  StrucId As String*4 'Structure identifier'
  Version As Long 'Structure version number'
  StrucLength As Long 'Total length of MQRFH2 including all'
  'NameValueLength and NameValueData fields'
  Encoding As Long 'Numeric encoding of data that follows'
  'last NameValueData field'
  CodedCharSetId As Long 'Character set identifier of data that'
  'follows last NameValueData field'
  Format As String*8 'Format name of data that follows last'
  'NameValueData field'
  Flags As Long 'Flags'
  NameValueCCSID As Long 'Character set identifier of NameValueData'
End Type
```

MQRFH2 – Language declarations

Chapter 18. MQRMH – Reference message header

The following table summarizes the fields in the structure.

Table 69. Fields in MQRMH

Field	Description	Page
<i>StrucId</i>	Structure identifier	277
<i>Version</i>	Structure version number	278
<i>StrucLength</i>	Total length of MQRMH, including strings at end of fixed fields, but not the bulk data	278
<i>Encoding</i>	Numeric encoding of bulk data	275
<i>CodedCharSetId</i>	Character set identifier of bulk data	273
<i>Format</i>	Format name of bulk data	276
<i>Flags</i>	Reference message flags	275
<i>ObjectType</i>	Object type	276
<i>ObjectInstanceId</i>	Object instance identifier	276
<i>SrcEnvLength</i>	Length of source environment data	276
<i>SrcEnvOffset</i>	Offset of source environment data	276
<i>SrcNameLength</i>	Length of source object name	277
<i>SrcNameOffset</i>	Offset of source object name	277
<i>DestEnvLength</i>	Length of destination environment data	274
<i>DestEnvOffset</i>	Offset of destination environment data	274
<i>DestNameLength</i>	Length of destination object name	274
<i>DestNameOffset</i>	Offset of destination object name	275
<i>DataLogicalLength</i>	Length of bulk data	273
<i>DataLogicalOffset</i>	Low offset of bulk data	273
<i>DataLogicalOffset2</i>	High offset of bulk data	274

Overview

Availability: AIX, HP-UX, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Purpose: The MQRMH structure defines the format of a reference message header. This header is used in conjunction with user-written message channel exits to send extremely large amounts of data (called “bulk data”) from one queue manager to another. The difference compared to normal messaging is that the bulk data is not stored on a queue; instead, only a *reference* to the bulk data is stored on the queue. This reduces the possibility of MQ resources being exhausted by a small number of extremely large messages.

Format name: MQFMT_REF_MSG_HEADER.

MQRMH – Reference message header

Character set and encoding: Character data in MQRMH, and the strings addressed by the offset fields, must be in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute. Numeric data in MQRMH must be in the native machine encoding; this is given by the value of `MQENC_NATIVE` for the C programming language.

The character set and encoding of the MQRMH must be set into the *CodedCharSetId* and *Encoding* fields in:

- The MQMD (if the MQRMH structure is at the start of the message data), or
- The header structure that precedes the MQRMH structure (all other cases).

Usage: An application puts a message consisting of an MQRMH, but omitting the bulk data. When the message is read from the transmission queue by a message channel agent (MCA), a user-supplied message exit is invoked to process the reference message header. The exit can append to the reference message the bulk data identified by the MQRMH structure, before the MCA sends the message through the channel to the next queue manager.

At the receiving end, a message exit that waits for reference messages should exist. When a reference message is received, the exit should create the object from the bulk data that follows the MQRMH in the message, and then pass on the reference message without the bulk data. The reference message can later be retrieved by an application reading the reference message (without the bulk data) from a queue.

Normally, the MQRMH structure is all that is in the message. However, if the message is on a transmission queue, one or more additional headers will precede the MQRMH structure.

A reference message can also be sent to a distribution list. In this case, the MQDH structure and its related records precede the MQRMH structure when the message is on a transmission queue.

Note: A reference message should not be sent as a segmented message, because the message exit cannot process it correctly.

Data conversion: For data conversion purposes, conversion of the MQRMH structure includes conversion of the source environment data, source object name, destination environment data, and destination object name. Any other bytes within *StrucLength* bytes of the start of the structure are either discarded or have undefined values after data conversion. The bulk data will be converted provided that all of the following are true:

- The bulk data is present in the message when the data conversion is performed.
- The *Format* field in MQRMH has a value other than `MQFMT_NONE`.
- A user-written data-conversion exit exists with the format name specified.

Be aware, however, that usually the bulk data is *not* present in the message when the message is on a queue, and that as a result the bulk data will not be converted by the `MQGMO_CONVERT` option.

Fields

The MQRMH structure contains the following fields; the fields are described in **alphabetic order**:

CodedCharSetId (MQLONG)

Character set identifier of bulk data.

This specifies the character set identifier of the bulk data; it does not apply to character data in the MQRMH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

MQCCSI_INHERIT

Inherit character-set identifier of this structure.

Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

MQCCSI_INHERIT cannot be used if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

This value is supported in the following environments: AIX, HP-UX, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

The initial value of this field is MQCCSI_UNDEFINED.

DataLogicalLength (MQLONG)

Length of bulk data.

The *DataLogicalLength* field specifies the length of the bulk data referenced by the MQRMH structure.

If the bulk data is actually present in the message, the data begins at an offset of *StrucLength* bytes from the start of the MQRMH structure. The length of the entire message minus *StrucLength* gives the length of the bulk data present.

If data is present in the message, *DataLogicalLength* specifies the amount of that data that is relevant. The normal case is for *DataLogicalLength* to have the same value as the length of data actually present in the message.

If the MQRMH structure represents the remaining data in the object (starting from the specified logical offset), the value zero can be used for *DataLogicalLength*, provided that the bulk data is not actually present in the message.

If no data is present, the end of MQRMH coincides with the end of the message.

The initial value of this field is 0.

DataLogicalOffset (MQLONG)

Low offset of bulk data.

This field specifies the low offset of the bulk data from the start of the object of which the bulk data forms part. The offset of the bulk data from the start of the object is called the *logical offset*. This is *not* the physical offset of the bulk data from the start of the MQRMH structure – that offset is given by *StrucLength*.

MQRMH – DataLogicalOffset field

To allow large objects to be sent using reference messages, the logical offset is divided into two fields, and the actual logical offset is given by the sum of these two fields:

- *DataLogicalOffset* represents the remainder obtained when the logical offset is divided by 1 000 000 000. It is thus a value in the range 0 through 999 999 999.
- *DataLogicalOffset2* represents the result obtained when the logical offset is divided by 1 000 000 000. It is thus the number of complete multiples of 1 000 000 000 that exist in the logical offset. The number of multiples is in the range 0 through 999 999 999.

The initial value of this field is 0.

DataLogicalOffset2 (MQLONG)

High offset of bulk data.

This field specifies the high offset of the bulk data from the start of the object of which the bulk data forms part. It is a value in the range 0 through 999 999 999. See *DataLogicalOffset* for details.

The initial value of this field is 0.

DestEnvLength (MQLONG)

Length of destination environment data.

If this field is zero, there is no destination environment data, and *DestEnvOffset* is ignored.

DestEnvOffset (MQLONG)

Offset of destination environment data.

This field specifies the offset of the destination environment data from the start of the MQRMH structure. Destination environment data can be specified by the creator of the reference message, if that data is known to the creator. For example, on OS/2 the destination environment data might be the directory path of the object where the bulk data is to be stored. However, if the creator does not know the destination environment data, it is the responsibility of the user-supplied message exit to determine any environment information needed.

The length of the destination environment data is given by *DestEnvLength*; if this length is zero, there is no destination environment data, and *DestEnvOffset* is ignored. If present, the destination environment data must reside completely within *StrucLength* bytes from the start of the structure.

Applications should not assume that the destination environment data is contiguous with any of the data addressed by the *SrcEnvOffset*, *SrcNameOffset*, and *DestNameOffset* fields.

The initial value of this field is 0.

DestNameLength (MQLONG)

Length of destination object name.

If this field is zero, there is no destination object name, and *DestNameOffset* is ignored.

DestNameOffset (MQLONG)

Offset of destination object name.

This field specifies the offset of the destination object name from the start of the MQRMH structure. The destination object name can be specified by the creator of the reference message, if that data is known to the creator. However, if the creator does not know the destination object name, it is the responsibility of the user-supplied message exit to identify the object to be created or modified.

The length of the destination object name is given by *DestNameLength*; if this length is zero, there is no destination object name, and *DestNameOffset* is ignored. If present, the destination object name must reside completely within *StrucLength* bytes from the start of the structure.

Applications should not assume that the destination object name is contiguous with any of the data addressed by the *SrcEnvOffset*, *SrcNameOffset*, and *DestEnvOffset* fields.

The initial value of this field is 0.

Encoding (MQLONG)

Numeric encoding of bulk data.

This specifies the numeric encoding of the bulk data; it does not apply to numeric data in the MQRMH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is MQENC_NATIVE.

Flags (MQLONG)

Reference message flags.

The following flags are defined:

MQRMHF_LAST

Reference message contains or represents last part of object.

This flag indicates that the reference message represents or contains the last part of the referenced object.

MQRMHF_NOT_LAST

Reference message does not contain or represent last part of object.

MQRMHF_NOT_LAST is defined to aid program documentation. It is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

The initial value of this field is MQRMHF_NOT_LAST.

MQRMH – Format field

Format (MQCHAR8)

Format name of bulk data.

This specifies the format name of the bulk data.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *Format* field in MQMD.

The initial value of this field is MQFMT_NONE.

ObjectInstanceld (MQBYTE24)

Object instance identifier.

This field can be used to identify a specific instance of an object. If it is not needed, it should be set to the following value:

MQOII_NONE

No object instance identifier specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQOII_NONE_ARRAY is also defined; this has the same value as MQOII_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_OBJECT_INSTANCE_ID_LENGTH. The initial value of this field is MQOII_NONE.

ObjectType (MQCHAR8)

Object type.

This is a name that can be used by the message exit to recognize types of reference message that it supports. It is recommended that the name conform to the same rules as the *Format* field described above.

The initial value of this field is 8 blanks.

SrcEnvLength (MQLONG)

Length of source environment data.

If this field is zero, there is no source environment data, and *SrcEnvOffset* is ignored.

The initial value of this field is 0.

SrcEnvOffset (MQLONG)

Offset of source environment data.

This field specifies the offset of the source environment data from the start of the MQRMH structure. Source environment data can be specified by the creator of the reference message, if that data is known to the creator. For example, on OS/2 the source environment data might be the directory path of the object containing the

bulk data. However, if the creator does not know the source environment data, it is the responsibility of the user-supplied message exit to determine any environment information needed.

The length of the source environment data is given by *SrcEnvLength*; if this length is zero, there is no source environment data, and *SrcEnvOffset* is ignored. If present, the source environment data must reside completely within *StrucLength* bytes from the start of the structure.

Applications should not assume that the environment data starts immediately after the last fixed field in the structure or that it is contiguous with any of the data addressed by the *SrcNameOffset*, *DestEnvOffset*, and *DestNameOffset* fields.

The initial value of this field is 0.

SrcNameLength (MQLONG)

Length of source object name.

If this field is zero, there is no source object name, and *SrcNameOffset* is ignored.

The initial value of this field is 0.

SrcNameOffset (MQLONG)

Offset of source object name.

This field specifies the offset of the source object name from the start of the MQRMH structure. The source object name can be specified by the creator of the reference message, if that data is known to the creator. However, if the creator does not know the source object name, it is the responsibility of the user-supplied message exit to identify the object to be accessed.

The length of the source object name is given by *SrcNameLength*; if this length is zero, there is no source object name, and *SrcNameOffset* is ignored. If present, the source object name must reside completely within *StrucLength* bytes from the start of the structure.

Applications should not assume that the source object name is contiguous with any of the data addressed by the *SrcEnvOffset*, *DestEnvOffset*, and *DestNameOffset* fields.

The initial value of this field is 0.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQRMH_STRUC_ID

Identifier for reference message header structure.

For the C programming language, the constant `MQRMH_STRUC_ID_ARRAY` is also defined; this has the same value as `MQRMH_STRUC_ID`, but is an array of characters instead of a string.

The initial value of this field is `MQRMH_STRUC_ID`.

MQRMH – StrucLength field

StrucLength (MQLONG)

Total length of MQRMH, including strings at end of fixed fields, but not the bulk data.

The initial value of this field is zero.

Version (MQLONG)

Structure version number.

The value must be:

MQRMH_VERSION_1

Version-1 reference message header structure.

The following constant specifies the version number of the current version:

MQRMH_CURRENT_VERSION

Current version of reference message header structure.

The initial value of this field is MQRMH_VERSION_1.

Initial values and language declarations

Table 70. Initial values of fields in MQRMH

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQRMH_STRUC_ID	'RMHb'
<i>Version</i>	MQRMH_VERSION_1	1
<i>StrucLength</i>	None	0
<i>Encoding</i>	MQENC_NATIVE	Depends on environment
<i>CodedCharSetId</i>	MQCCSI_UNDEFINED	0
<i>Format</i>	MQFMT_NONE	Blanks
<i>Flags</i>	MQRMHF_NOT_LAST	0
<i>ObjectType</i>	None	Blanks
<i>ObjectInstanceId</i>	MQOIL_NONE	Nulls
<i>SrcEnvLength</i>	None	0
<i>SrcEnvOffset</i>	None	0
<i>SrcNameLength</i>	None	0
<i>SrcNameOffset</i>	None	0
<i>DestEnvLength</i>	None	0
<i>DestEnvOffset</i>	None	0
<i>DestNameLength</i>	None	0
<i>DestNameOffset</i>	None	0
<i>DataLogicalLength</i>	None	0
<i>DataLogicalOffset</i>	None	0
<i>DataLogicalOffset2</i>	None	0

Table 70. Initial values of fields in MQRMH (continued)

Field name	Name of constant	Value of constant
Notes:		
1. The symbol 'b' represents a single blank character.		
2. In the C programming language, the macro variable MQRMH_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:		
MQRMH MyRMH = {MQRMH_DEFAULT};		

C declaration

```
typedef struct tagMQRMH MQRMH;
struct tagMQRMH {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQLONG    StrucLength;      /* Total length of MQRMH, including
                               strings at end of fixed fields, but
                               not the bulk data */
    MQLONG    Encoding;         /* Numeric encoding of bulk data */
    MQLONG    CodedCharSetId;   /* Character set identifier of bulk
                               data */
    MQCHAR8   Format;           /* Format name of bulk data */
    MQLONG    Flags;            /* Reference message flags */
    MQCHAR8   ObjectType;       /* Object type */
    MQBYTE24  ObjectInstanceId; /* Object instance identifier */
    MQLONG    SrcEnvLength;     /* Length of source environment data */
    MQLONG    SrcEnvOffset;     /* Offset of source environment data */
    MQLONG    SrcNameLength;    /* Length of source object name */
    MQLONG    SrcNameOffset;    /* Offset of source object name */
    MQLONG    DestEnvLength;    /* Length of destination environment
                               data */
    MQLONG    DestEnvOffset;    /* Offset of destination environment
                               data */
    MQLONG    DestNameLength;   /* Length of destination object name */
    MQLONG    DestNameOffset;   /* Offset of destination object name */
    MQLONG    DataLogicalLength; /* Length of bulk data */
    MQLONG    DataLogicalOffset; /* Low offset of bulk data */
    MQLONG    DataLogicalOffset2; /* High offset of bulk data */
};
```

COBOL declaration

```
** MQRMH structure
10 MQRMH.
** Structure identifier
15 MQRMH-STRUCID PIC X(4).
** Structure version number
15 MQRMH-VERSION PIC S9(9) BINARY.
** Total length of MQRMH, including strings at end of fixed fields,
** but not the bulk data
15 MQRMH-STRUCLength PIC S9(9) BINARY.
** Numeric encoding of bulk data
15 MQRMH-ENCODING PIC S9(9) BINARY.
** Character set identifier of bulk data
15 MQRMH-CODEDCHARSETID PIC S9(9) BINARY.
** Format name of bulk data
15 MQRMH-FORMAT PIC X(8).
** Reference message flags
15 MQRMH-FLAGS PIC S9(9) BINARY.
** Object type
15 MQRMH-OBJECTTYPE PIC X(8).
```

MQRMH – Language declarations

```
** Object instance identifier
15 MQRMH-OBJECTINSTANCEID PIC X(24).
** Length of source environment data
15 MQRMH-SRCENVLENGTH PIC S9(9) BINARY.
** Offset of source environment data
15 MQRMH-SRCENVOFFSET PIC S9(9) BINARY.
** Length of source object name
15 MQRMH-SRCNAMELENGTH PIC S9(9) BINARY.
** Offset of source object name
15 MQRMH-SRCNAMEOFFSET PIC S9(9) BINARY.
** Length of destination environment data
15 MQRMH-DESTENVLENGTH PIC S9(9) BINARY.
** Offset of destination environment data
15 MQRMH-DESTENVOFFSET PIC S9(9) BINARY.
** Length of destination object name
15 MQRMH-DESTNAMELENGTH PIC S9(9) BINARY.
** Offset of destination object name
15 MQRMH-DESTNAMEOFFSET PIC S9(9) BINARY.
** Length of bulk data
15 MQRMH-DATALOGICALENGTH PIC S9(9) BINARY.
** Low offset of bulk data
15 MQRMH-DATALOGICALOFFSET PIC S9(9) BINARY.
** High offset of bulk data
15 MQRMH-DATALOGICALOFFSET2 PIC S9(9) BINARY.
```

PL/I declaration

```
dc1
1 MQRMH based,
3 StrucId char(4), /* Structure identifier */
3 Version fixed bin(31), /* Structure version number */
3 StrucLength fixed bin(31), /* Total length of MQRMH,
including strings at end of
fixed fields, but not the bulk
data */
3 Encoding fixed bin(31), /* Numeric encoding of bulk
data */
3 CodedCharSetId fixed bin(31), /* Character set identifier of
bulk data */
3 Format char(8), /* Format name of bulk data */
3 Flags fixed bin(31), /* Reference message flags */
3 ObjectType char(8), /* Object type */
3 ObjectInstanceId char(24), /* Object instance identifier */
3 SrcEnvLength fixed bin(31), /* Length of source environment
data */
3 SrcEnvOffset fixed bin(31), /* Offset of source environment
data */
3 SrcNameLength fixed bin(31), /* Length of source object name */
3 SrcNameOffset fixed bin(31), /* Offset of source object name */
3 DestEnvLength fixed bin(31), /* Length of destination
environment data */
3 DestEnvOffset fixed bin(31), /* Offset of destination
environment data */
3 DestNameLength fixed bin(31), /* Length of destination object
name */
3 DestNameOffset fixed bin(31), /* Offset of destination object
name */
3 DataLogicalLength fixed bin(31), /* Length of bulk data */
3 DataLogicalOffset fixed bin(31), /* Low offset of bulk data */
3 DataLogicalOffset2 fixed bin(31); /* High offset of bulk data */
```

System/390 assembler declaration

```
MQRMH DSECT
MQRMH_STRUCID DS CL4 Structure identifier
MQRMH_VERSION DS F Structure version number
MQRMH_STRUCLNGTH DS F Total length of MQRMH, including
```

MQRMH – Language declarations

```

*           strings at end of fixed fields, but
*           not the bulk data
MQRMH_ENCODING      DS  F  Numeric encoding of bulk data
MQRMH_CODEDCHARSETID DS  F  Character set identifier of bulk
*           data
MQRMH_FORMAT        DS  CL8 Format name of bulk data
MQRMH_FLAGS         DS  F  Reference message flags
MQRMH_OBJECTTYPE    DS  CL8 Object type
MQRMH_OBJECTINSTANCEID DS XL24 Object instance identifier
MQRMH_SRCENVLENGTH  DS  F  Length of source environment data
MQRMH_SRCENVOFFSET  DS  F  Offset of source environment data
MQRMH_SRCNAMELENGTH DS  F  Length of source object name
MQRMH_SRCNAMEOFFSET DS  F  Offset of source object name
MQRMH_DESTENVLENGTH DS  F  Length of destination environment
*           data
MQRMH_DESTENVOFFSET DS  F  Offset of destination environment
*           data
MQRMH_DESTNAMELENGTH DS  F  Length of destination object name
MQRMH_DESTNAMEOFFSET DS  F  Offset of destination object name
MQRMH_DATALOGICALENGTH DS  F  Length of bulk data
MQRMH_DATALOGICALOFFSET DS  F  Low offset of bulk data
MQRMH_DATALOGICALOFFSET2 DS  F  High offset of bulk data
*
MQRMH_LENGTH        EQU  *-MQRMH
                    ORG  MQRMH
MQRMH_AREA          DS   CL(MQRMH_LENGTH)

```

Visual Basic declaration

```

Type MQRMH
  StrucId      As String*4 'Structure identifier'
  Version      As Long     'Structure version number'
  StrucLength  As Long     'Total length of MQRMH, including'
                    'strings at end of fixed fields, but'
                    'not the bulk data'
  Encoding     As Long     'Numeric encoding of bulk data'
  CodedCharSetId As Long   'Character set identifier of bulk data'
  Format        As String*8 'Format name of bulk data'
  Flags        As Long     'Reference message flags'
  ObjectType    As String*8 'Object type'
  ObjectInstanceId As MQBYTE24 'Object instance identifier'
  SrcEnvLength  As Long     'Length of source environment data'
  SrcEnvOffset  As Long     'Offset of source environment data'
  SrcNameLength As Long     'Length of source object name'
  SrcNameOffset As Long     'Offset of source object name'
  DestEnvLength As Long     'Length of destination environment'
                    'data'
  DestEnvOffset As Long     'Offset of destination environment'
                    'data'
  DestNameLength As Long     'Length of destination object name'
  DestNameOffset As Long     'Offset of destination object name'
  DataLogicalLength As Long   'Length of bulk data'
  DataLogicalOffset As Long   'Low offset of bulk data'
  DataLogicalOffset2 As Long  'High offset of bulk data'
End Type

```

MQRMH – Language declarations

Chapter 19. MQRR – Response record

The following table summarizes the fields in the structure.

Table 71. Fields in MQRR

Field	Description	Page
<i>CompCode</i>	Completion code for queue	283
<i>Reason</i>	Reason code for queue	283

Overview

Availability: AIX, HP-UX, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Purpose: The MQRR structure is used to receive the completion code and reason code resulting from the open or put operation for a single destination queue, when the destination is a distribution list. MQRR is an output structure for the MQOPEN, MQPUT, and MQPUT1 calls.

Character set and encoding: Data in MQRR must be in the character set and encoding of the local queue manager; these are given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE, respectively. However, if the application is running as an MQ client, the structure must be in the character set and encoding of the client.

Usage: By providing an array of these structures on the MQOPEN and MQPUT calls, or on the MQPUT1 call, it is possible to determine the completion codes and reason codes for all of the queues in a distribution list when the outcome of the call is mixed, that is, when the call succeeds for some queues in the list but fails for others. Reason code MQRC_MULTIPLE_REASONS from the call indicates that the response records (if provided by the application) have been set by the queue manager.

Fields

The MQRR structure contains the following fields; the fields are described in **alphabetic order**:

CompCode (MQLONG)

Completion code for queue.

This is the completion code resulting from the open or put operation for the queue whose name was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call.

This is always an output field. The initial value of this field is MQCC_OK.

Reason (MQLONG)

Reason code for queue.

MQRR – Reason field

This is the reason code resulting from the open or put operation for the queue whose name was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call.

This is always an output field. The initial value of this field is MQRC_NONE.

Initial values and language declarations

Table 72. Initial values of fields in MQRR

Field name	Name of constant	Value of constant
CompCode	MQCC_OK	0
Reason	MQRC_NONE	0

Notes:

1. In the C programming language, the macro variable MQRR_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQRR MyRR = {MQRR_DEFAULT};
```

C declaration

```
typedef struct tagMQRR MQRR;  
struct tagMQRR {  
    MQLONG CompCode; /* Completion code for queue */  
    MQLONG Reason; /* Reason code for queue */  
};
```

COBOL declaration

```
** MQRR structure  
10 MQRR.  
** Completion code for queue  
15 MQRR-COMPCODE PIC S9(9) BINARY.  
** Reason code for queue  
15 MQRR-REASON PIC S9(9) BINARY.
```

PL/I declaration

```
dcl  
1 MQRR based,  
3 CompCode fixed bin(31), /* Completion code for queue */  
3 Reason fixed bin(31); /* Reason code for queue */
```

Visual Basic declaration

```
Type MQRR  
CompCode As Long 'Completion code for queue'  
Reason As Long 'Reason code for queue'  
End Type
```

Chapter 20. MQSCO – SSL configuration options

The following table summarizes the fields in the structure.

Table 73. Fields in MQSCO

Field	Description	Page
<i>StrucId</i>	Structure identifier	288
<i>Version</i>	Structure version number	288
<i>KeyRepository</i>	Location of key repository	287
<i>CryptoHardware</i>	Details of cryptographic hardware	286
<i>AuthInfoRecCount</i>	Number of MQAIR records present	285
<i>AuthInfoRecOffset</i>	Offset of first MQAIR record from start of MQSCO	286
<i>AuthInfoRecPtr</i>	Address of first MQAIR record	286

Overview

Availability: AIX, HP-UX, Solaris, Linux and Windows clients, apart from Windows 98 clients.

Purpose: The MQSCO structure (in conjunction with the SSL fields in the MQCD structure) allows an application running as a WebSphere MQ client to specify configuration options that control the use of SSL for the client connection when the channel protocol is TCP/IP. The structure is an input parameter on the MQCONN call.

If the channel protocol for the client channel is not TCP/IP, the MQSCO structure is ignored.

Character set and encoding: Data in MQSCO must be in the character set and encoding of the local queue manager; these are given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE, respectively.

Fields

The MQSCO structure contains the following fields; the fields are described in **alphabetic order**:

AuthInfoRecCount (MQLONG)

Number of MQAIR records present; see Chapter 2, “MQAIR – Authentication information record”, on page 31.

This is the number of authentication information records addressed by the *AuthInfoRecPtr* or *AuthInfoRecOffset* fields. The value must be zero or greater. If the value is not valid, the call fails with reason code MQRC_AUTH_INFO_REC_COUNT_ERROR.

This is an input field. The initial value of this field is 0.

MQSCO – AuthInfoRecOffset field

AuthInfoRecOffset (MQLONG)

Offset of first MQAIR record from start of MQSCO structure.

This is the offset in bytes of the first authentication information record from the start of the MQSCO structure. The offset can be positive or negative. The field is ignored if *AuthInfoRecCount* is zero.

Either *AuthInfoRecOffset* or *AuthInfoRecPtr* can be used to specify the MQAIR records, but not both; see the description of the *AuthInfoRecPtr* field for details.

This is an input field. The initial value of this field is 0.

AuthInfoRecPtr (PMQAIR)

Address of first MQAIR record.

This is the address of the first authentication information record. The field is ignored if *AuthInfoRecCount* is zero.

The array of MQAIR records can be provided in one of two ways:

- By using the pointer field *AuthInfoRecPtr*

In this case, the application can declare an array of MQAIR records that is separate from the MQSCO structure, and set *AuthInfoRecPtr* to the address of the array.

Using *AuthInfoRecPtr* is recommended for programming languages that support the pointer data type in a fashion that is portable to different environments (for example, the C programming language).

- By using the offset field *AuthInfoRecOffset*

In this case, the application should declare a compound structure containing an MQSCO followed by the array of MQAIR records, and set *AuthInfoRecOffset* to the offset of the first record in the array from the start of the MQSCO structure. Care must be taken to ensure that this value is correct, and has a value that can be accommodated within an MQLONG (the most restrictive programming language is COBOL, for which the valid range is -999 999 999 through +999 999 999).

Using *AuthInfoRecOffset* is recommended for programming languages that do not support the pointer data type, or that implement the pointer data type in a fashion that is not portable to different environments (for example, the COBOL programming language).

Whichever technique is chosen, only one of *AuthInfoRecPtr* and *AuthInfoRecOffset* can be used; the call fails with reason code MQRC_AUTH_INFO_REC_ERROR if both are nonzero.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise.

Note: On platforms where the programming language does not support the pointer datatype, this field is declared as a byte string of the appropriate length.

CryptoHardware (MQCHAR256)

Cryptographic hardware configuration string.

This field is relevant only for WebSphere MQ clients running on UNIX systems. It gives configuration details for cryptographic hardware connected to the client system. The field must be set to one of the following strings, or left blank or null:

```
GSK_ACCELERATOR_RAINBOW_CS_OFF
GSK_ACCELERATOR_RAINBOW_CS_ON
GSK_ACCELERATOR_NCIPHER_NF_OFF
GSK_ACCELERATOR_NCIPHER_NF_ON
GSK_PKCS11=PKCS#11 driver path; PKCS#11 token label;PKCS#11 token password
```

Notes:

1. The strings containing RAINBOW enable or disable the Rainbow cryptographic hardware (it is enabled by default if it is present).
2. The strings containing NCIPHER enable or disable the nCipher cryptographic hardware (it is enabled by default if it is present).
3. In order to use cryptographic hardware which conforms to the PKCS11 interface, for example, the IBM 4960 or IBM 4963, the PKCS11 driver path, PKCS11 token label, and PKCS11 token password strings must be specified, each terminated by a semi-colon.
4. If the field is blank or null, it indicates that no cryptographic hardware configuration is required.

If the value is shorter than the length of the field, the value must be terminated by a null character, or padded with blanks to the length of the field. If the value is not valid, or leads to a failure when used to configure the cryptographic hardware, the call fails with reason code MQRC_CRYPTO_HARDWARE_ERROR.

This is an input field. The length of this field is given by MQ_SSL_CRYPTO_HARDWARE_LENGTH. The initial value of this field is the null string in C, and blank characters in other programming languages.

KeyRepository (MQCHAR256)

Location of SSL key repository.

This field is relevant only for WebSphere MQ clients running on UNIX systems and Windows systems. It specifies the location of the key database file in which keys and certificates are stored. On UNIX systems the key database file must have a file name of the form *zzz.kdb*, where *zzz* is user-selectable. The *KeyRepository* field contains the path to this file, along with the file name stem (all characters in the file name up to but not including the final *.kdb*). The *.kdb* file suffix is added automatically.

Each key database file has an associated “password stash file”. This holds encrypted passwords that are used to allow programmatic access to the key database. The password stash file must reside in the same directory and have the same file stem as the key database, and must end with the suffix *.sth*.

For example, if the *KeyRepository* field has the value */xxx/yyy/key*, the key database file must be */xxx/yyy/key.kdb*, and the password stash file must be */xxx/yyy/key.sth*, where *xxx* and *yyy* represent directory names.

If the value is shorter than the length of the field, the value must be terminated by a null character, or padded with blanks to the length of the field. The value is not checked; if there is an error in accessing the key repository, the call fails with reason code MQRC_KEY_REPOSITORY_ERROR.

MQSCO – KeyRepository field

In order to run an SSL connection from a WebSphere MQ client, on UNIX systems, *KeyRepository* must be set to a valid key database file name.

This is an input field. The length of this field is given by `MQ_SSL_KEY_REPOSITORY_LENGTH`. The initial value of this field is the null string in C, and blank characters in other programming languages.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQSCO_STRUC_ID

Identifier for SSL configuration options structure.

For the C programming language, the constant `MQSCO_STRUC_ID_ARRAY` is also defined; this has the same value as `MQSCO_STRUC_ID`, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is `MQSCO_STRUC_ID`.

Version (MQLONG)

Structure version number.

The value must be:

MQSCO_VERSION_1

Version-1 SSL configuration options structure.

The following constant specifies the version number of the current version:

MQSCO_CURRENT_VERSION

Current version of SSL configuration options structure.

This is always an input field. The initial value of this field is `MQSCO_VERSION_1`.

Initial values and language declarations

Table 74. Initial values of fields in MQSCO

Field name	Name of constant	Value of constant
<i>StrucId</i>	<code>MQSCO_STRUC_ID</code>	'SC0b'
<i>Version</i>	<code>MQSCO_CURRENT_VERSION</code>	1
<i>KeyRepository</i>	None	Null string or blanks
<i>CryptoHardware</i>	None	Null string or blanks
<i>AuthInfoRecCount</i>	None	0
<i>AuthInfoRecOffset</i>	None	0
<i>AuthInfoRecPtr</i>	None	Null pointer or null bytes

Table 74. Initial values of fields in MQSCO (continued)

Field name	Name of constant	Value of constant
Notes:		
1. The symbol 'b' represents a single blank character.		
2. In the C programming language, the macro variable MQSCO_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:		
MQSCO MySCO = {MQSCO_DEFAULT};		

C declaration

```
typedef struct tagMQSCO MQSCO;
struct tagMQSCO {
    MQCHAR4    StrucId;           /* Structure identifier */
    MQLONG     Version;          /* Structure version number */
    MQCHAR256  KeyRepository;    /* Location of SSL key repository */
    MQCHAR256  CryptoHardware;   /* Cryptographic hardware configuration
                                string */
    MQLONG     AuthInfoRecCount; /* Number of MQAIR records present */
    MQLONG     AuthInfoRecOffset; /* Offset of first MQAIR record from
                                start of MQSCO structure */
    PMQAIR     AuthInfoRecPtr;   /* Address of first MQAIR record */
};
```

COBOL declaration

```
** MQSCO structure
10 MQSCO.
** Structure identifier
15 MQSCO-STRUCID          PIC X(4).
** Structure version number
15 MQSCO-VERSION         PIC S9(9) BINARY.
** Location of SSL key repository
15 MQSCO-KEYREPOSITORY   PIC X(256).
** Cryptographic hardware configuration string
15 MQSCO-CRYPTOHardware PIC X(256).
** Number of MQAIR records present
15 MQSCO-AUTHINFORECCOUNT PIC S9(9) BINARY.
** Offset of first MQAIR record from start of MQSCO structure
15 MQSCO-AUTHINFORECOFFSET PIC S9(9) BINARY.
** Address of first MQAIR record
15 MQSCO-AUTHINFORECPtr  POINTER.
```

PL/I declaration

```
dc1
1 MQSCO based,
3 StrucId          char(4),      /* Structure identifier */
3 Version          fixed bin(31), /* Structure version number */
3 KeyRepository    char(256),    /* Location of SSL key
                                repository */
3 CryptoHardware   char(256),    /* Cryptographic hardware
                                configuration string */
3 AuthInfoRecCount fixed bin(31), /* Number of MQAIR records
                                present */
3 AuthInfoRecOffset fixed bin(31), /* Offset of first MQAIR record
                                from start of MQSCO structure */
3 AuthInfoRecPtr   pointer;      /* Address of first MQAIR record */
```

MQSCO – Language declarations

Visual Basic declaration

```
|
|
|      Type MQSCO
|      StrucId      As String*4  'Structure identifier'
|      Version      As Long      'Structure version number'
|      KeyRepository As String*256 'Location of SSL key repository'
|      CryptoHardware As String*256 'Cryptographic hardware configuration'
|
|      AuthInfoRecCount As Long  'Number of MQAIR records present'
|      AuthInfoRecOffset As Long  'Offset of first MQAIR record from'
|
|      AuthInfoRecPtr   As MQPTR  'start of MQSCO structure'
|      'Address of first MQAIR record'
|
|      End Type
```

Chapter 21. MQTM – Trigger message

The following table summarizes the fields in the structure.

Table 75. Fields in MQTM

Field	Description	Page
<i>StrucId</i>	Structure identifier	295
<i>Version</i>	Structure version number	296
<i>QName</i>	Name of triggered queue	295
<i>ProcessName</i>	Name of process object	294
<i>TriggerData</i>	Trigger data	295
<i>ApplType</i>	Application type	293
<i>ApplId</i>	Application identifier	293
<i>EnvData</i>	Environment data	294
<i>UserData</i>	User data	296

Overview

Purpose: The MQTM structure describes the data in the trigger message that is sent by the queue manager to a trigger-monitor application when a trigger event occurs for a queue. This structure is part of the WebSphere MQ Trigger Monitor Interface (TMI), which is one of the WebSphere MQ framework interfaces.

Format name: MQFMT_TRIGGER.

Character set and encoding: Character data in MQTM is in the character set of the queue manager that generates the MQTM. Numeric data in MQTM is in the machine encoding of the queue manager that generates the MQTM.

The character set and encoding of the MQTM are given by the *CodedCharSetId* and *Encoding* fields in:

- The MQMD (if the MQTM structure is at the start of the message data), or
- The header structure that precedes the MQTM structure (all other cases).

Usage: A trigger-monitor application may need to pass some or all of the information in the trigger message to the application which is started by the trigger-monitor application. Information which may be needed by the started application includes *QName*, *TriggerData*, and *UserData*. The trigger-monitor application can pass the MQTM structure directly to the started application, or pass an MQTMC2 structure instead, depending on what is permitted by the environment and convenient for the started application. For information about MQTMC2, see Chapter 22, “MQTMC2 – Trigger message 2 (character format)”, on page 299.

- On z/OS, for an MQAT_CICS application that is started using the CKTI transaction, the entire trigger message structure MQTM is made available to the started transaction; the information can be retrieved by using the EXEC CICS RETRIEVE command.

MQTM – Trigger message

- On OS/400, the trigger-monitor application provided with WebSphere MQ passes an MQTMC2 structure to the started application.
- On VSE/ESA, triggered programs are invoked by the queue manager using either the transaction ID code or the program ID code in the queue definition. This transaction ID or program ID determines if the trigger is invoked using an EXEC CICS START or an EXEC CICS LINK.
 - Triggered programs invoked using the START mechanism can use EXEC CICS RETRIEVE to retrieve the MQTM structure.
 - Triggered programs invoked using the LINK mechanism can retrieve the MQTM structure in the DFH COMMAREA.
- On Windows 3.1, Windows 95, Windows 98, there is no trigger-monitor application, and this structure is not supported.

For information about triggers, see the *WebSphere MQ Application Programming Guide*.

MQMD for a trigger message: The fields in the MQMD of a trigger message generated by the queue manager are set as follows:

Field in MQMD	Value used
<i>StrucId</i>	MQMD_STRUC_ID
<i>Version</i>	MQMD_VERSION_1
<i>Report</i>	MQRO_NONE
<i>MsgType</i>	MQMT_DATAGRAM
<i>Expiry</i>	MQEI_UNLIMITED
<i>Feedback</i>	MQFB_NONE
<i>Encoding</i>	MQENC_NATIVE
<i>CodedCharSetId</i>	Queue manager's <i>CodedCharSetId</i> attribute
<i>Format</i>	MQFMT_TRIGGER
<i>Priority</i>	Initiation queue's <i>DefPriority</i> attribute
<i>Persistence</i>	MQPER_NOT_PERSISTENT
<i>MsgId</i>	A unique value
<i>CorrelId</i>	MQCI_NONE
<i>BackoutCount</i>	0
<i>ReplyToQ</i>	Blanks
<i>ReplyToQMgr</i>	Name of queue manager
<i>UserIdentifier</i>	Blanks
<i>AccountingToken</i>	MQACT_NONE
<i>ApplIdentityData</i>	Blanks
<i>PutApplType</i>	MQAT_QMGR, or as appropriate for the message channel agent
<i>PutApplName</i>	First 28 bytes of the queue-manager name
<i>PutDate</i>	Date when trigger message is sent
<i>PutTime</i>	Time when trigger message is sent
<i>ApplOriginData</i>	Blanks

An application that generates a trigger message is recommended to set similar values, except for the following:

- The *Priority* field can be set to MQPRI_PRIORITY_AS_Q_DEF (the queue manager will change this to the default priority for the initiation queue when the message is put).
- The *ReplyToQMgr* field can be set to blanks (the queue manager will change this to the name of the local queue manager when the message is put).
- The context fields should be set as appropriate for the application.

Fields

The MQTM structure contains the following fields; the fields are described in **alphabetic order**:

AppId (MQCHAR256)

Application identifier.

This is a character string that identifies the application to be started, and is used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the *AppId* attribute of the process object identified by the *ProcessName* field; see Chapter 42, “Attributes for process definitions”, on page 495 for details of this attribute. The content of this data is of no significance to the queue manager.

The meaning of *AppId* is determined by the trigger-monitor application. The trigger monitor provided by WebSphere MQ requires *AppId* to be the name of an executable program. The following notes apply to the environments indicated:

- On z/OS, *AppId* is:
 - A CICS transaction identifier, for applications started using the CICS trigger-monitor transaction CKTI
 - An IMS transaction identifier, for applications started using the IMS trigger monitor CSQQTRMN
- On PC DOS, OS/2, and Windows systems, the program name may be prefixed with a drive and directory path.
- On OS/400, the program name may be prefixed with a library name and / character.
- On UNIX systems, the program name may be prefixed with a directory path.
- On VSE/ESA, *AppId* is a CICS transaction identifier.

The length of this field is given by MQ_PROCESS_APPL_ID_LENGTH. The initial value of this field is the null string in C, and 256 blank characters in other programming languages.

AppType (MQLONG)

Application type.

This identifies the nature of the program to be started, and is used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the *AppType* attribute of the process object identified by the *ProcessName* field; see Chapter 42, “Attributes for process definitions”, on page 495 for details of this attribute. The content of this data is of no significance to the queue manager.

AppType can have one of the following standard values. User-defined types can also be used, but should be restricted to values in the range MQAT_USER_FIRST through MQAT_USER_LAST:

MQAT_AIX

AIX application (same value as MQAT_UNIX).

MQAT_CICS

CICS transaction.

MQAT_CICS_VSE

CICS/VSE transaction.

MQTM – ApplType field

MQAT_DOS	WebSphere MQ client application on PC DOS.
MQAT_IMS	IMS application.
MQAT_MVS	MVS or TSO application (same value as MQAT_ZOS).
MQAT_NOTES_AGENT	Lotus Notes Agent application.
MQAT_NSK	Compaq NonStop Kernel application.
MQAT_OS2	OS/2 or Presentation Manager application.
MQAT_OS390	OS/390 application (same value as MQAT_ZOS).
MQAT_OS400	OS/400 application.
MQAT_UNIX	UNIX application.
MQAT_VMS	Digital OpenVMS application.
MQAT_WINDOWS	16-bit Windows application.
MQAT_WINDOWS_NT	32-bit Windows application.
MQAT_WLM	z/OS workload manager application.
MQAT_ZOS	z/OS application.
MQAT_USER_FIRST	Lowest value for user-defined application type.
MQAT_USER_LAST	Highest value for user-defined application type.

The initial value of this field is 0.

EnvData (MQCHAR128)

Environment data.

This is a character string that contains environment-related information pertaining to the application to be started, and is used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the *EnvData* attribute of the process object identified by the *ProcessName* field; see Chapter 42, “Attributes for process definitions”, on page 495 for details of this attribute. The content of this data is of no significance to the queue manager.

On z/OS, for a CICS application started using the CKTI transaction, or an IMS application to be started using the CSQQTRMN transaction, this information is not used.

The length of this field is given by MQ_PROCESS_ENV_DATA_LENGTH. The initial value of this field is the null string in C, and 128 blank characters in other programming languages.

ProcessName (MQCHAR48)

Name of process object.

This is the name of the queue-manager process object specified for the triggered queue, and can be used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the *ProcessName* attribute of the queue identified by the *QName* field; see Chapter 40, “Attributes for queues”, on page 457 for details of this attribute.

Names that are shorter than the defined length of the field are always padded to the right with blanks; they are not ended prematurely by a null character.

The length of this field is given by `MQ_PROCESS_NAME_LENGTH`. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

QName (MQCHAR48)

Name of triggered queue.

This is the name of the queue for which a trigger event occurred, and is used by the application started by the trigger-monitor application. The queue manager initializes this field with the value of the *QName* attribute of the triggered queue; see Chapter 40, “Attributes for queues”, on page 457 for details of this attribute.

Names that are shorter than the defined length of the field are padded to the right with blanks; they are not ended prematurely by a null character.

The length of this field is given by `MQ_Q_NAME_LENGTH`. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQTM_STRUC_ID

Identifier for trigger message structure.

For the C programming language, the constant `MQTM_STRUC_ID_ARRAY` is also defined; this has the same value as `MQTM_STRUC_ID`, but is an array of characters instead of a string.

The initial value of this field is `MQTM_STRUC_ID`.

TriggerData (MQCHAR64)

Trigger data.

This is free-format data for use by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the *TriggerData* attribute of the queue identified by the *QName* field; see Chapter 40, “Attributes for queues”, on page 457 for details of this attribute. The content of this data is of no significance to the queue manager.

- On z/OS, for a CICS application started using the CKTI transaction, this information is not used.
- On VSE/ESA, this field is set as follows:
 - The first four bytes are set to the transaction ID code.
 - The next eight bytes are set to the program ID code.

MQTM – TriggerData field

- The next byte is set to the trigger event flag character, either 'F' for MQTT_FIRST, or 'E' for MQTT_EVERY.
- The remaining bytes are set to blanks.

The length of this field is given by MQ_TRIGGER_DATA_LENGTH. The initial value of this field is the null string in C, and 64 blank characters in other programming languages.

UserData (MQCHAR128)

User data.

This is a character string that contains user information relevant to the application to be started, and is used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the *UserData* attribute of the process object identified by the *ProcessName* field; see Chapter 42, “Attributes for process definitions”, on page 495 for details of this attribute. The content of this data is of no significance to the queue manager.

The length of this field is given by MQ_PROCESS_USER_DATA_LENGTH. The initial value of this field is the null string in C, and 128 blank characters in other programming languages.

Version (MQLONG)

Structure version number.

The value must be:

MQTM_VERSION_1

Version number for trigger message structure.

The following constant specifies the version number of the current version:

MQTM_CURRENT_VERSION

Current version of trigger message structure.

The initial value of this field is MQTM_VERSION_1.

Initial values and language declarations

Table 76. Initial values of fields in MQTM

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQTM_STRUC_ID	'Tmbb'
<i>Version</i>	MQTM_VERSION_1	1
<i>QName</i>	None	Null string or blanks
<i>ProcessName</i>	None	Null string or blanks
<i>TriggerData</i>	None	Null string or blanks
<i>ApplType</i>	None	0
<i>ApplId</i>	None	Null string or blanks
<i>EnvData</i>	None	Null string or blanks
<i>UserData</i>	None	Null string or blanks

Table 76. Initial values of fields in MQTM (continued)

Field name	Name of constant	Value of constant
Notes:		
1. The symbol 'b' represents a single blank character.		
2. The value 'Null string or blanks' denotes the null string in C, and blank characters in other programming languages.		
3. In the C programming language, the macro variable MQTM_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:		
MQTM MyTM = {MQTM_DEFAULT};		

C declaration

```
typedef struct tagMQTM MQTM;
struct tagMQTM {
    MQCHAR4    StrucId;        /* Structure identifier */
    MQLONG     Version;       /* Structure version number */
    MQCHAR48   QName;        /* Name of triggered queue */
    MQCHAR48   ProcessName;  /* Name of process object */
    MQCHAR64   TriggerData;  /* Trigger data */
    MQLONG     ApplType;     /* Application type */
    MQCHAR256  ApplId;       /* Application identifier */
    MQCHAR128  EnvData;      /* Environment data */
    MQCHAR128  UserData;    /* User data */
};
```

COBOL declaration

```
** MQTM structure
10 MQTM.
** Structure identifier
15 MQTM-STRUCID PIC X(4).
** Structure version number
15 MQTM-VERSION PIC S9(9) BINARY.
** Name of triggered queue
15 MQTM-QNAME PIC X(48).
** Name of process object
15 MQTM-PROCESSNAME PIC X(48).
** Trigger data
15 MQTM-TRIGGERDATA PIC X(64).
** Application type
15 MQTM-APPLTYPE PIC S9(9) BINARY.
** Application identifier
15 MQTM-APPLID PIC X(256).
** Environment data
15 MQTM-ENVDATA PIC X(128).
** User data
15 MQTM-USERDATA PIC X(128).
```

PL/I declaration

```
dcl
1 MQTM based,
3 StrucId char(4), /* Structure identifier */
3 Version fixed bin(31), /* Structure version number */
3 QName char(48), /* Name of triggered queue */
3 ProcessName char(48), /* Name of process object */
3 TriggerData char(64), /* Trigger data */
3 ApplType fixed bin(31), /* Application type */
```

MQTM – Language declarations

```
3 AppId      char(256),    /* Application identifier */
3 EnvData    char(128),    /* Environment data */
3 UserData   char(128);    /* User data */
```

System/390 assembler declaration

```
MQTM          DSECT
MQTM_STRUCID  DS  CL4      Structure identifier
MQTM_VERSION  DS  F        Structure version number
MQTM_QNAME    DS  CL48     Name of triggered queue
MQTM_PROCESSNAME DS  CL48  Name of process object
MQTM_TRIGGERDATA DS  CL64  Trigger data
MQTM_APPLTYPE DS  F        Application type
MQTM_APPLID   DS  CL256    Application identifier
MQTM_ENVDATA  DS  CL128    Environment data
MQTM_USERDATA DS  CL128    User data
*
MQTM_LENGTH   EQU  *-MQTM
              ORG  MQTM
MQTM_AREA     DS  CL(MQTM_LENGTH)
```

TAL declaration

```
STRUCT      MQTM^DEF (*);
BEGIN
STRUCT      STRUCID;
BEGIN STRING BYTE [0:3]; END;
INT(32)     VERSION;
STRUCT      QNAME;
BEGIN STRING BYTE [0:47]; END;
STRUCT      PROCESSNAME;
BEGIN STRING BYTE [0:47]; END;
STRUCT      TRIGGERDATA;
BEGIN STRING BYTE [0:63]; END;
INT(32)     APPLTYPE;
STRUCT      APPLID;
BEGIN STRING BYTE [0:255]; END;
STRUCT      ENVDATA;
BEGIN STRING BYTE [0:127]; END;
STRUCT      USERDATA;
BEGIN STRING BYTE [0:127]; END;
END;
```

Visual Basic declaration

```
Type MQTM
  StrucId As String*4 'Structure identifier'
  Version As Long     'Structure version number'
  QName As String*48  'Name of triggered queue'
  ProcessName As String*48 'Name of process object'
  TriggerData As String*64 'Trigger data'
  ApplType As Long    'Application type'
  ApplId As String*256 'Application identifier'
  EnvData As String*128 'Environment data'
  UserData As String*128 'User data'
End Type
```

Chapter 22. MQTMC2 – Trigger message 2 (character format)

The following table summarizes the fields in the structure.

Table 77. Fields in MQTMC2

Field	Description	Page
<i>StrucId</i>	Structure identifier	300
<i>Version</i>	Structure version number	301
<i>QName</i>	Name of triggered queue	300
<i>ProcessName</i>	Name of process object	300
<i>TriggerData</i>	Trigger data	300
<i>ApplType</i>	Application type	300
<i>ApplId</i>	Application identifier	300
<i>EnvData</i>	Environment data	300
<i>UserData</i>	User data	301
<i>QMgrName</i>	Queue manager name	300

Overview

Purpose: When a trigger-monitor application retrieves a trigger message (MQTM) from an initiation queue, the trigger monitor may need to pass some or all of the information in the trigger message to the application that is started by the trigger monitor. Information that may be needed by the started application includes *QName*, *TriggerData*, and *UserData*. The trigger monitor application can pass the MQTM structure directly to the started application, or pass an MQTMC2 structure instead, depending on what is permitted by the environment and convenient for the started application.

This structure is part of the WebSphere MQ Trigger Monitor Interface (TMI), which is one of the WebSphere MQ framework interfaces.

Character set and encoding: Character data in MQTMC2 is in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute.

Usage: The MQTMC2 structure is very similar to the format of the MQTM structure. The difference is that the non-character fields in MQTM are changed in MQTMC2 to character fields of the same length, and the queue manager name is added at the end of the structure.

- On z/OS, for an MQAT_IMS application that is started using the CSQQTRMN application, an MQTMC2 structure is made available to the started application.
- On OS/400, the trigger monitor application provided with WebSphere MQ passes an MQTMC2 structure to the started application.
- On VSE/ESA, this structure is not supported.
- On Windows 3.1, Windows 95, Windows 98, there is no trigger monitor application, and this structure is not supported.

Fields

The MQTMC2 structure contains the following fields; the fields are described in **alphabetic order**:

AppId (MQCHAR256)

Application identifier.

See the *AppId* field in the MQTM structure.

AppType (MQCHAR4)

Application type.

This field always contains blanks, whatever the value in the *AppType* field in the MQTM structure of the original trigger message.

EnvData (MQCHAR128)

Environment data.

See the *EnvData* field in the MQTM structure.

ProcessName (MQCHAR48)

Name of process object.

See the *ProcessName* field in the MQTM structure.

QMgrName (MQCHAR48)

Queue manager name.

This is the name of the queue manager at which the trigger event occurred.

QName (MQCHAR48)

Name of triggered queue.

See the *QName* field in the MQTM structure.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQTMC_STRUC_ID

Identifier for trigger message (character format) structure.

For the C programming language, the constant `MQTMC_STRUC_ID_ARRAY` is also defined; this has the same value as `MQTMC_STRUC_ID`, but is an array of characters instead of a string.

TriggerData (MQCHAR64)

Trigger data.

See the *TriggerData* field in the MQTM structure.

UserData (MQCHAR128)

User data.

See the *UserData* field in the MQTM structure.

Version (MQCHAR4)

Structure version number.

The value must be:

MQTMC_VERSION_2

Version 2 trigger message (character format) structure.

For the C programming language, the constant MQTMC_VERSION_2_ARRAY is also defined; this has the same value as MQTMC_VERSION_2, but is an array of characters instead of a string.

The following constant specifies the version number of the current version:

MQTMC_CURRENT_VERSION

Current version of trigger message (character format) structure.

Initial values and language declarations

Table 78. Initial values of fields in MQTMC2

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQTMC_STRUC_ID	'TMCb'
<i>Version</i>	MQTMC_VERSION_2	'bbb2'
<i>QName</i>	None	Null string or blanks
<i>ProcessName</i>	None	Null string or blanks
<i>TriggerData</i>	None	Null string or blanks
<i>ApplType</i>	None	Blanks
<i>ApplId</i>	None	Null string or blanks
<i>EnvData</i>	None	Null string or blanks
<i>UserData</i>	None	Null string or blanks
<i>QMgrName</i>	None	Null string or blanks
Notes:		
1. The symbol 'b' represents a single blank character.		
2. The value 'Null string or blanks' denotes the null string in C, and blank characters in other programming languages.		
3. In the C programming language, the macro variable MQTMC2_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:		
MQTMC2 MyTMC = {MQTMC2_DEFAULT};		

C declaration

```
typedef struct tagMQTMC2 MQTMC2;
struct tagMQTMC2 {
    MQCHAR4    StrucId;    /* Structure identifier */
```

MQTMC2 – Language declarations

```
MQCHAR4    Version;      /* Structure version number */
MQCHAR48   QName;       /* Name of triggered queue */
MQCHAR48   ProcessName; /* Name of process object */
MQCHAR64   TriggerData; /* Trigger data */
MQCHAR4    ApplType;    /* Application type */
MQCHAR256  ApplId;      /* Application identifier */
MQCHAR128  EnvData;     /* Environment data */
MQCHAR128  UserData;    /* User data */
MQCHAR48   QMgrName;    /* Queue manager name */
};
```

COBOL declaration

```
** MQTMC2 structure
10 MQTMC2.
** Structure identifier
15 MQTMC2-STRUCID PIC X(4).
** Structure version number
15 MQTMC2-VERSION PIC X(4).
** Name of triggered queue
15 MQTMC2-QNAME PIC X(48).
** Name of process object
15 MQTMC2-PROCESSNAME PIC X(48).
** Trigger data
15 MQTMC2-TRIGGERDATA PIC X(64).
** Application type
15 MQTMC2-APPLTYPE PIC X(4).
** Application identifier
15 MQTMC2-APPLID PIC X(256).
** Environment data
15 MQTMC2-ENVDATA PIC X(128).
** User data
15 MQTMC2-USERDATA PIC X(128).
** Queue manager name
15 MQTMC2-QMGRNAME PIC X(48).
```

PL/I declaration

```
dc1
1 MQTMC2 based,
3 StrucId char(4), /* Structure identifier */
3 Version char(4), /* Structure version number */
3 QName char(48), /* Name of triggered queue */
3 ProcessName char(48), /* Name of process object */
3 TriggerData char(64), /* Trigger data */
3 ApplType char(4), /* Application type */
3 ApplId char(256), /* Application identifier */
3 EnvData char(128), /* Environment data */
3 UserData char(128), /* User data */
3 QMgrName char(48); /* Queue manager name */
```

System/390 assembler declaration

```
MQTMC          DSECT
MQTMC_STRUCID  DS CL4 Structure identifier
MQTMC_VERSION  DS CL4 Structure version number
MQTMC_QNAME    DS CL48 Name of triggered queue
MQTMC_PROCESSNAME DS CL48 Name of process object
MQTMC_TRIGGERDATA DS CL64 Trigger data
MQTMC_APPLTYPE DS CL4 Application type
MQTMC_APPLID   DS CL256 Application identifier
MQTMC_ENVDATA  DS CL128 Environment data
MQTMC_USERDATA DS CL128 User data
MQTMC_QMGRNAME DS CL48 Queue manager name
```

```

*
MQTMC_LENGTH      EQU  *-MQTMC
                   ORG  MQTMC
MQTMC_AREA        DS   CL(MQTMC_LENGTH)

```

TAL declaration

```

STRUCT      MQTMC2^DEF (*);
BEGIN
STRUCT      STRUCID;
BEGIN STRING BYTE [0:3]; END;
STRUCT      VERSION;
BEGIN STRING BYTE [0:3]; END;
STRUCT      QNAME;
BEGIN STRING BYTE [0:47]; END;
STRUCT      PROCESSNAME;
BEGIN STRING BYTE [0:47]; END;
STRUCT      TRIGGERDATA;
BEGIN STRING BYTE [0:63]; END;
STRUCT      APPLTYPE;
BEGIN STRING BYTE [0:3]; END;
STRUCT      APPLID;
BEGIN STRING BYTE [0:255]; END;
STRUCT      ENVDATA;
BEGIN STRING BYTE [0:127]; END;
STRUCT      USERDATA;
BEGIN STRING BYTE [0:127]; END;
STRUCT      QMQRNAME;
BEGIN STRING BYTE [0:47]; END;
END;

```

Visual Basic declaration

```

Type MQTMC2
  StrucId   As String*4   'Structure identifier'
  Version   As String*4   'Structure version number'
  QName     As String*48  'Name of triggered queue'
  ProcessName As String*48 'Name of process object'
  TriggerData As String*64 'Trigger data'
  ApplType  As String*4   'Application type'
  ApplId    As String*256 'Application identifier'
  EnvData   As String*128 'Environment data'
  UserData  As String*128 'User data'
  QMgrName  As String*48  'Queue manager name'
End Type

```

MQTMC2 – Language declarations

Chapter 23. MQWIH – Work information header

The following table summarizes the fields in the structure.

Table 79. Fields in MQWIH

Field	Description	Page
<i>StrucId</i>	Structure identifier	307
<i>Version</i>	Structure version number	308
<i>StrucLength</i>	Length of MQWIH structure	307
<i>Encoding</i>	Numeric encoding of data that follows MQWIH	306
<i>CodedCharSetId</i>	Character-set identifier of data that follows MQWIH	305
<i>Format</i>	Format name of data that follows MQWIH	306
<i>Flags</i>	Flags	306
<i>ServiceName</i>	Service name	307
<i>ServiceStep</i>	Service step name	307
<i>MsgToken</i>	Message token	307
<i>Reserved</i>	Reserved	307

Overview

Availability: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Purpose: The MQWIH structure describes the information that must be present at the start of a message that is to be handled by the z/OS workload manager.

Format name: MQFMT_WORK_INFO_HEADER.

Character set and encoding: The fields in the MQWIH structure are in the character set and encoding given by the *CodedCharSetId* and *Encoding* fields in the header structure that precedes MQWIH, or by those fields in the MQMD structure if the MQWIH is at the start of the application message data.

The character set must be one that has single-byte characters for the characters that are valid in queue names.

Usage: If a message is to be processed by the z/OS workload manager, the message must begin with an MQWIH structure.

Fields

The MQWIH structure contains the following fields; the fields are described in **alphabetic order**:

CodedCharSetId (MQLONG)

Character-set identifier of data that follows MQWIH.

MQWIH – CodedCharSetId field

This specifies the character set identifier of the data that follows the MQWIH structure; it does not apply to character data in the MQWIH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

MQCCSI_INHERIT

Inherit character-set identifier of this structure.

Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

MQCCSI_INHERIT cannot be used if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

This value is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG)

Numeric encoding of data that follows MQWIH.

This specifies the numeric encoding of the data that follows the MQWIH structure; it does not apply to numeric data in the MQWIH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is 0.

Flags (MQLONG)

Flags

The value must be:

MQWIH_NONE

No flags.

The initial value of this field is MQWIH_NONE.

Format (MQCHAR8)

Format name of data that follows MQWIH.

This specifies the format name of the data that follows the MQWIH structure.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *Format* field in MQMD.

The length of this field is given by `MQ_FORMAT_LENGTH`. The initial value of this field is `MQFMT_NONE`.

MsgToken (MQBYTE16)

Message token.

This is a message token that uniquely identifies the message.

For the `MQPUT` and `MQPUT1` calls, this field is ignored. The length of this field is given by `MQ_MSG_TOKEN_LENGTH`. The initial value of this field is `MQMTOK_NONE`.

Reserved (MQCHAR32)

Reserved.

This is a reserved field; it must be blank.

ServiceName (MQCHAR32)

Service name.

This is the name of the service that is to process the message.

The length of this field is given by `MQ_SERVICE_NAME_LENGTH`. The initial value of this field is 32 blank characters.

ServiceStep (MQCHAR8)

Service step name.

This is the name of the step of *ServiceName* to which the message relates.

The length of this field is given by `MQ_SERVICE_STEP_LENGTH`. The initial value of this field is 8 blank characters.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQWIH_STRUC_ID

Identifier for work information header structure.

For the C programming language, the constant `MQWIH_STRUC_ID_ARRAY` is also defined; this has the same value as `MQWIH_STRUC_ID`, but is an array of characters instead of a string.

The initial value of this field is `MQWIH_STRUC_ID`.

StrucLength (MQLONG)

Length of MQWIH structure.

The value must be:

MQWIH_LENGTH_1

Length of version-1 work information header structure.

MQWIH – StrucLength field

The following constant specifies the length of the current version:

MQWIH_CURRENT_LENGTH

Length of current version of work information header structure.

The initial value of this field is MQWIH_LENGTH_1.

Version (MQLONG)

Structure version number.

The value must be:

MQWIH_VERSION_1

Version-1 work information header structure.

The following constant specifies the version number of the current version:

MQWIH_CURRENT_VERSION

Current version of work information header structure.

The initial value of this field is MQWIH_VERSION_1.

Initial values and language declarations

Table 80. Initial values of fields in MQWIH

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQWIH_STRUC_ID	'WIHb'
<i>Version</i>	MQWIH_VERSION_1	1
<i>StrucLength</i>	MQWIH_LENGTH_1	120
<i>Encoding</i>	None	0
<i>CodedCharSetId</i>	MQCCSI_UNDEFINED	0
<i>Format</i>	MQFMT_NONE	Blanks
<i>Flags</i>	MQWIH_NONE	0
<i>ServiceName</i>	None	Blanks
<i>ServiceStep</i>	None	Blanks
<i>MsgToken</i>	MQMTOK_NONE	Nulls
<i>Reserved</i>	None	Blanks
Notes: <ol style="list-style-type: none">1. The symbol 'b' represents a single blank character.2. In the C programming language, the macro variable MQWIH_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure: <pre>MQWIH MyWIH = {MQWIH_DEFAULT};</pre>		

C declaration

```
typedef struct tagMQWIH MQWIH;  
struct tagMQWIH {  
    MQCHAR4   StrucId;        /* Structure identifier */  
    MQLONG    Version;       /* Structure version number */  
};
```

MQWIH – Language declarations

```
MQLONG   StruLength;    /* Length of MQWIH structure */
MQLONG   Encoding;     /* Numeric encoding of data that follows
                        MQWIH */
MQLONG   CodedCharSetId; /* Character-set identifier of data that
                        follows MQWIH */
MQCHAR8   Format;      /* Format name of data that follows
                        MQWIH */
MQLONG   Flags;       /* Flags */
MQCHAR32  ServiceName; /* Service name */
MQCHAR8   ServiceStep; /* Service step name */
MQBYTE16  MsgToken;   /* Message token */
MQCHAR32  Reserved;   /* Reserved */
};
```

COBOL declaration

```
** MQWIH structure
10 MQWIH.
** Structure identifier
15 MQWIH-STRUCID PIC X(4).
** Structure version number
15 MQWIH-VERSION PIC S9(9) BINARY.
** Length of MQWIH structure
15 MQWIH-STRUCLength PIC S9(9) BINARY.
** Numeric encoding of data that follows MQWIH
15 MQWIH-ENCODING PIC S9(9) BINARY.
** Character-set identifier of data that follows MQWIH
15 MQWIH-CODEDCHARSETID PIC S9(9) BINARY.
** Format name of data that follows MQWIH
15 MQWIH-FORMAT PIC X(8).
** Flags
15 MQWIH-FLAGS PIC S9(9) BINARY.
** Service name
15 MQWIH-SERVICENAME PIC X(32).
** Service step name
15 MQWIH-SERVICESTEP PIC X(8).
** Message token
15 MQWIH-MSGTOKEN PIC X(16).
** Reserved
15 MQWIH-RESERVED PIC X(32).
```

PL/I declaration

```
dc1
1 MQWIH based,
3 StrucId char(4), /* Structure identifier */
3 Version fixed bin(31), /* Structure version number */
3 StruLength fixed bin(31), /* Length of MQWIH structure */
3 Encoding fixed bin(31), /* Numeric encoding of data that
                        follows MQWIH */
3 CodedCharSetId fixed bin(31), /* Character-set identifier of data
                        that follows MQWIH */
3 Format char(8), /* Format name of data that follows
                        MQWIH */
3 Flags fixed bin(31), /* Flags */
3 ServiceName char(32), /* Service name */
3 ServiceStep char(8), /* Service step name */
3 MsgToken char(16), /* Message token */
3 Reserved char(32); /* Reserved */
```

System/390 assembler declaration

```
MQWIH DSECT
MQWIH_STRUCID DS CL4 Structure identifier
MQWIH_VERSION DS F Structure version number
MQWIH_STRUCLength DS F Length of MQWIH structure
MQWIH_ENCODING DS F Numeric encoding of data that follows
```

MQWIH – Language declarations

```
*
MQWIH_CODEDCHARSETID DS F MQWIH
                        Character-set identifier of data that
*                        follows MQWIH
MQWIH_FORMAT          DS CL8 Format name of data that follows MQWIH
MQWIH_FLAGS           DS F   Flags
MQWIH_SERVICENAME     DS CL32 Service name
MQWIH_SERVICESTEP     DS CL8  Service step name
MQWIH_MSGTOKEN        DS XL16 Message token
MQWIH_RESERVED        DS CL32 Reserved
*
MQWIH_LENGTH          EQU *-MQWIH
                        ORG MQWIH
MQWIH_AREA            DS CL(MQWIH_LENGTH)
```

Visual Basic declaration

```
Type MQWIH
  StrucId      As String*4 'Structure identifier'
  Version      As Long     'Structure version number'
  StrucLength  As Long     'Length of MQWIH structure'
  Encoding     As Long     'Numeric encoding of data that follows'
  CodedCharSetId As Long   'MQWIH'
  CodedCharSetId As Long   'Character-set identifier of data that'
  CodedCharSetId As Long   'follows MQWIH'
  Format       As String*8 'Format name of data that follows MQWIH'
  Flags       As Long     'Flags'
  ServiceName As String*32 'Service name'
  ServiceStep As String*8  'Service step name'
  MsgToken    As MQBYTE16 'Message token'
  Reserved    As String*32 'Reserved'
End Type
```

Chapter 24. MQXP – Exit parameter block

The following table summarizes the fields in the structure.

Table 81. Fields in MQXP

Field	Description	Page
<i>StrucId</i>	Structure identifier	314
<i>Version</i>	Structure version number	314
<i>ExitId</i>	Exit identifier	312
<i>ExitReason</i>	Reason for invocation of exit	312
<i>ExitResponse</i>	Response from exit	313
<i>ExitCommand</i>	API call code	311
<i>ExitParmCount</i>	Parameter count	312
<i>ExitUserArea</i>	User area	313

Overview

Availability: z/OS.

Purpose: The MQXP structure is used as an input/output parameter to the API-crossing exit. For more information on this exit, see the *WebSphere MQ Application Programming Guide*.

Character set and encoding: Character data in MQXP is in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute. Numeric data in MQXP is in the native machine encoding; this is given by MQENC_NATIVE.

Fields

The MQXP structure contains the following fields; the fields are described in **alphabetic order**:

ExitCommand (MQLONG)

API call code.

This field is set on entry to the exit routine. It identifies the API call that caused the exit to be invoked:

MQXC_MQBACK
The MQBACK call.

MQXC_MQCLOSE
The MQCLOSE call.

MQXC_MQCMIT
The MQCMIT call.

MQXC_MQGET
The MQGET call.

MQXP – ExitCommand field

MQXC_MQINQ
The MQINQ call.

MQXC_MQOPEN
The MQOPEN call.

MQXC_MQPUT
The MQPUT call.

MQXC_MQPUT1
The MQPUT1 call.

MQXC_MQSET
The MQSET call.

This is an input field to the exit.

ExitId (MQLONG)

Exit identifier.

This is set on entry to the exit routine, and indicates the type of exit:

MQXT_API_CROSSING_EXIT
API-crossing exit for CICS.

This is an input field to the exit.

ExitParmCount (MQLONG)

Parameter count.

This field is set on entry to the exit routine. It contains the number of parameters that the MQ call takes. These are:

Call name	Number of parameters
MQBACK	3
MQCLOSE	5
MQCMIT	3
MQGET	9
MQINQ	10
MQOPEN	6
MQPUT	8
MQPUT1	8
MQSET	10

This is an input field to the exit.

ExitReason (MQLONG)

Reason for invocation of exit.

This is set on entry to the exit routine. For the API-crossing exit it indicates whether the routine is called before or after execution of the API call:

MQXR_BEFORE
Before API execution.

MQXR_AFTER
After API execution.

This is an input field to the exit.

ExitResponse (MQLONG)

Response from exit.

The value is set by the exit to communicate with the caller. The following values are defined:

MQXCC_OK

Exit completed successfully.

MQXCC_SUPPRESS_FUNCTION

Suppress function.

When this value is set by an API-crossing exit called *before* the API call, the API call is not performed. The *CompCode* for the call is set to MQCC_FAILED, the *Reason* is set to MQRC_SUPPRESSED_BY_EXIT, and all other parameters remain as the exit left them.

When this value is set by an API-crossing exit called *after* the API call, it is ignored by the queue manager.

MQXCC_SKIP_FUNCTION

Skip function.

When this value is set by an API-crossing exit called *before* the API call, the API call is not performed; the *CompCode* and *Reason* and all other parameters remain as the exit left them.

When this value is set by an API-crossing exit called *after* the API call, it is ignored by the queue manager.

This is an output field from the exit.

ExitUserArea (MQBYTE16)

User area.

This is a field that is available for the exit to use. It is initialized to binary zero for the length of the field before the first invocation of the exit for the task, and thereafter any changes made to this field by the exit are preserved across invocations of the exit. The following value is defined:

MQXUA_NONE

No user information.

The value is binary zero for the length of the field.

For the C programming language, the constant MQXUA_NONE_ARRAY is also defined; this has the same value as MQXUA_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_EXIT_USER_AREA_LENGTH. This is an input/output field to the exit.

Reserved (MQLONG)

Reserved.

This is a reserved field. Its value is not significant to the exit.

MQXP – StructId field

StructId (MQCHAR4)

Structure identifier.

The value must be:

MQXP_STRUC_ID

Identifier for exit parameter structure.

For the C programming language, the constant MQXP_STRUC_ID_ARRAY is also defined; this has the same value as MQXP_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the exit.

Version (MQLONG)

Structure version number.

The value must be:

MQXP_VERSION_1

Version number for exit parameter-block structure.

Note: When a new version of this structure is introduced, the layout of the existing part is not changed. The exit should therefore check that the version number is equal to or greater than the lowest version that contains the fields that the exit needs to use.

This is an input field to the exit.

Language declarations

This structure is supported in the following programming languages.

C declaration

```
typedef struct tagMQXP MQXP;
struct tagMQXP {
    MQCHAR4   StructId;           /* Structure identifier */
    MQLONG    Version;           /* Structure version number */
    MQLONG    ExitId;           /* Exit identifier */
    MQLONG    ExitReason;       /* Reason for invocation of exit */
    MQLONG    ExitResponse;     /* Response from exit */
    MQLONG    ExitCommand;     /* API call code */
    MQLONG    ExitParmCount;    /* Parameter count */
    MQLONG    Reserved;        /* Reserved */
    MQBYTE16  ExitUserArea;     /* User area */
};
```

COBOL declaration

```
** MQXP structure
10 MQXP.
** Structure identifier
15 MQXP-STRUCID PIC X(4).
** Structure version number
15 MQXP-VERSION PIC S9(9) BINARY.
** Exit identifier
15 MQXP-EXITID PIC S9(9) BINARY.
** Reason for invocation of exit
15 MQXP-EXITREASON PIC S9(9) BINARY.
** Response from exit
```

```

15 MQXP-EXITRESPONSE PIC S9(9) BINARY.
**   API call code
15 MQXP-EXITCOMMAND PIC S9(9) BINARY.
**   Parameter count
15 MQXP-EXITPARMCOUNT PIC S9(9) BINARY.
**   Reserved
15 MQXP-RESERVED PIC S9(9) BINARY.
**   User area
15 MQXP-EXITUSERAREA PIC X(16).

```

PL/I declaration

```

dcl
  1 MQXP based,
    3 StrucId      char(4),      /* Structure identifier */
    3 Version      fixed bin(31), /* Structure version number */
    3 ExitId       fixed bin(31), /* Exit identifier */
    3 ExitReason   fixed bin(31), /* Reason for invocation of exit */
    3 ExitResponse fixed bin(31), /* Response from exit */
    3 ExitCommand  fixed bin(31), /* API call code */
    3 ExitParmCount fixed bin(31), /* Parameter count */
    3 Reserved     fixed bin(31), /* Reserved */
    3 ExitUserArea char(16);    /* User area */

```

System/390 assembler declaration

```

MQXP          DSECT
MQXP_STRUCID  DS   CL4  Structure identifier
MQXP_VERSION  DS   F    Structure version number
MQXP_EXITID   DS   F    Exit identifier
MQXP_EXITREASON DS   F    Reason for invocation of exit
MQXP_EXITRESPONSE DS   F    Response from exit
MQXP_EXITCOMMAND DS   F    API call code
MQXP_EXITPARMCOUNT DS   F    Parameter count
MQXP_RESERVED DS   F    Reserved
MQXP_EXITUSERAREA DS   XL16 User area
*
MQXP_LENGTH   EQU   *-MQXP
              ORG   MQXP
MQXP_AREA     DS   CL(MQXP_LENGTH)

```

MQXP – Language declarations

Chapter 25. MQXQH – Transmission-queue header

The following table summarizes the fields in the structure.

Table 82. Fields in MQXQH

Field	Description	Page
<i>StrucId</i>	Structure identifier	321
<i>Version</i>	Structure version number	321
<i>RemoteQName</i>	Name of destination queue	321
<i>RemoteQMgrName</i>	Name of destination queue manager	320
<i>MsgDesc</i>	Original message descriptor	320

Overview

Availability: All.

Note: Message channels are not available when using the reduced function form of WebSphere MQ for z/OS supplied with WebSphere Application Server.

Purpose: The MQXQH structure describes the information that is prefixed to the application message data of messages when they are on transmission queues. A transmission queue is a special type of local queue that temporarily holds messages destined for remote queues (that is, destined for queues that do not belong to the local queue manager). A transmission queue is denoted by the *Usage* queue attribute having the value MQUS_TRANSMISSION.

Format name: MQFMT_XMIT_Q_HEADER.

Character set and encoding: Data in MQXQH must be in the character set and encoding of the local queue manager; these are given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE for the C programming language, respectively.

The character set and encoding of the MQXQH must be set into the *CodedCharSetId* and *Encoding* fields in:

- The separate MQMD (if the MQXQH structure is at the start of the message data), or
- The header structure that precedes the MQXQH structure (all other cases).

Usage: A message that is on a transmission queue has *two* message descriptors:

- One message descriptor is stored separately from the message data; this is called the *separate message descriptor*, and is generated by the queue manager when the message is placed on the transmission queue. Some of the fields in the separate message descriptor are copied from the message descriptor provided by the application on the MQPUT or MQPUT1 call (see below for details).

The separate message descriptor is the one that is returned to the application in the *MsgDesc* parameter of the MQGET call when the message is removed from the transmission queue.

MQXQH – Transmission-queue header

- A second message descriptor is stored within the MQXQH structure as part of the message data; this is called the *embedded message descriptor*, and is a copy of the message descriptor that was provided by the application on the MQPUT or MQPUT1 call (with minor variations – see below for details).

The embedded message descriptor is always a version-1 MQMD. If the message put by the application has nondefault values for one or more of the version-2 fields in the MQMD, an MQMDE structure follows the MQXQH, and is in turn followed by the application message data (if any). The MQMDE is either:

- Generated by the queue manager (if the application uses a version-2 MQMD to put the message), or
- Already present at the start of the application message data (if the application uses a version-1 MQMD to put the message).

The embedded message descriptor is the one that is returned to the application in the *MsgDesc* parameter of the MQGET call when the message is removed from the final destination queue.

Fields in the separate message descriptor: The fields in the separate message descriptor are set by the queue manager as shown below. If the queue manager does not support the version-2 MQMD, a version-1 MQMD is used without loss of function.

Field in separate MQMD	Value used
<i>StrucId</i>	MQMD_STRUC_ID
<i>Version</i>	MQMD_VERSION_2
<i>Report</i>	Copied from the embedded message descriptor, but with the bits identified by MQRO_ACCEPT_UNSUP_IF_XMIT_MASK set to zero. (This prevents a COA or COD report message being generated when a message is placed on or removed from a transmission queue.)
<i>MsgType</i>	Copied from the embedded message descriptor.
<i>Expiry</i>	Copied from the embedded message descriptor.
<i>Feedback</i>	Copied from the embedded message descriptor.
<i>Encoding</i>	MQENC_NATIVE (see note below)
<i>CodedCharSetId</i>	Queue manager's <i>CodedCharSetId</i> attribute.
<i>Format</i>	MQFMT_XMIT_Q_HEADER
<i>Priority</i>	Copied from the embedded message descriptor.
<i>Persistence</i>	Copied from the embedded message descriptor.
<i>MsgId</i>	A new value is generated by the queue manager. This message identifier is different from the <i>MsgId</i> that the queue manager may have generated for the embedded message descriptor (see above).
<i>CorrelId</i>	The <i>MsgId</i> from the embedded message descriptor.
<i>BackoutCount</i>	0
<i>ReplyToQ</i>	Copied from the embedded message descriptor.
<i>ReplyToQMGr</i>	Copied from the embedded message descriptor.
<i>UserIdentifier</i>	Copied from the embedded message descriptor.
<i>AccountingToken</i>	Copied from the embedded message descriptor.
<i>ApplIdentityData</i>	Copied from the embedded message descriptor.
<i>PutApplType</i>	MQAT_QMGR
<i>PutApplName</i>	First 28 bytes of the queue-manager name.
<i>PutDate</i>	Date when message was put on transmission queue.
<i>PutTime</i>	Time when message was put on transmission queue.
<i>ApplOriginData</i>	Blanks
<i>GroupId</i>	MQGL_NONE

MQXQH – Transmission-queue header

Field in separate MQMD	Value used
<i>MsgSeqNumber</i>	1
<i>Offset</i>	0
<i>MsgFlags</i>	MQMF_NONE
<i>OriginalLength</i>	MQOL_UNDEFINED

- On OS/2 and Windows, the value of MQENC_NATIVE for Micro Focus COBOL differs from the value for C. The value in the *Encoding* field in the separate message descriptor is always the value for C in these environments; this value is 546 in decimal. Also, the integer fields in the MQXQH structure are in the encoding that corresponds to this value (the native Intel encoding).

Fields in the embedded message descriptor: The fields in the embedded message descriptor have the same values as those in the *MsgDesc* parameter of the MQPUT or MQPUT1 call, with the exception of the following:

- The *Version* field always has the value MQMD_VERSION_1.
- If the *Priority* field has the value MQPRI_PRIORITY_AS_Q_DEF, it is replaced by the value of the queue's *DefPriority* attribute.
- If the *Persistence* field has the value MQPER_PERSISTENCE_AS_Q_DEF, it is replaced by the value of the queue's *DefPersistence* attribute.
- If the *MsgId* field has the value MQMI_NONE, or the MQPMO_NEW_MSG_ID option was specified, or the message is a distribution-list message, *MsgId* is replaced by a new message identifier generated by the queue manager.
When a distribution-list message is split into smaller distribution-list messages placed on different transmission queues, the *MsgId* field in each of the new embedded message descriptors is the same as that in the original distribution-list message.
- If the MQPMO_NEW_CORREL_ID option was specified, *CorrelId* is replaced by a new correlation identifier generated by the queue manager.
- The context fields are set as indicated by the MQPMO_*_CONTEXT options specified in the *PutMsgOpts* parameter; the context fields are:
 - AccountingToken*
 - ApplIdentityData*
 - ApplOriginData*
 - PutApplName*
 - PutApplType*
 - PutDate*
 - PutTime*
 - UserIdentifier*
- The version-2 fields (if they were present) are removed from the MQMD, and moved into an MQMDE structure, if one or more of the version-2 fields has a nondefault value.

Putting messages on remote queues: When an application puts a message on a remote queue (either by specifying the name of the remote queue directly, or by using a local definition of the remote queue), the local queue manager:

- Creates an MQXQH structure containing the embedded message descriptor
- Appends an MQMDE if one is needed and is not already present
- Appends the application message data
- Places the message on an appropriate transmission queue

Putting messages directly on transmission queues: It is also possible for an application to put a message directly on a transmission queue. In this case the application must prefix the application message data with an MQXQH structure,

MQXQH – Transmission-queue header

and initialize the fields with appropriate values. In addition, the *Format* field in the *MsgDesc* parameter of the MQPUT or MQPUT1 call must have the value MQFMT_XMIT_Q_HEADER.

Character data in the MQXQH structure created by the application must be in the character set of the local queue manager (defined by the *CodedCharSetId* queue-manager attribute), and integer data must be in the native machine encoding. In addition, character data in the MQXQH structure must be padded with blanks to the defined length of the field; the data must not be ended prematurely by using a null character, because the queue manager does not convert the null and subsequent characters to blanks in the MQXQH structure.

Note however that the queue manager does not check that an MQXQH structure is present, or that valid values have been specified for the fields.

Getting messages from transmission queues: Applications that get messages from a transmission queue must process the information in the MQXQH structure in an appropriate fashion. The presence of the MQXQH structure at the beginning of the application message data is indicated by the value MQFMT_XMIT_Q_HEADER being returned in the *Format* field in the *MsgDesc* parameter of the MQGET call. The values returned in the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter indicate the character set and encoding of the character and integer data in the MQXQH structure, respectively. The character set and encoding of the application message data are defined by the *CodedCharSetId* and *Encoding* fields in the embedded message descriptor.

Fields

The MQXQH structure contains the following fields; the fields are described in **alphabetic order**:

MsgDesc (MQMD1)

Original message descriptor.

This is the embedded message descriptor, and is a close copy of the message descriptor MQMD that was specified as the *MsgDesc* parameter on the MQPUT or MQPUT1 call when the message was originally put to the remote queue.

Note: This is a version-1 MQMD.

The initial values of the fields in this structure are the same as those in the MQMD structure.

RemoteQMgrName (MQCHAR48)

Name of destination queue manager.

This is the name of the queue manager or queue-sharing group that owns the queue that is the apparent eventual destination for the message.

If the message is a distribution-list message, *RemoteQMgrName* is blank.

The length of this field is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

RemoteQName (MQCHAR48)

Name of destination queue.

This is the name of the message queue that is the apparent eventual destination for the message (this may prove not to be the actual eventual destination if, for example, this queue is defined at *RemoteQMgrName* to be a local definition of another remote queue).

If the message is a distribution-list message (that is, the *Format* field in the embedded message descriptor is MQFMT_DIST_HEADER), *RemoteQName* is blank.

The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQXQH_STRUC_ID

Identifier for transmission-queue header structure.

For the C programming language, the constant MQXQH_STRUC_ID_ARRAY is also defined; this has the same value as MQXQH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQXQH_STRUC_ID.

Version (MQLONG)

Structure version number.

The value must be:

MQXQH_VERSION_1

Version number for transmission-queue header structure.

The following constant specifies the version number of the current version:

MQXQH_CURRENT_VERSION

Current version of transmission-queue header structure.

The initial value of this field is MQXQH_VERSION_1.

Initial values and language declarations

Table 83. Initial values of fields in MQXQH

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQXQH_STRUC_ID	'XQHb'
<i>Version</i>	MQXQH_VERSION_1	1
<i>RemoteQName</i>	None	Null string or blanks
<i>RemoteQMgrName</i>	None	Null string or blanks
<i>MsgDesc</i>	Same names and values as MQMD; see Table 52 on page 197	–

MQXQH – Language declarations

Table 83. Initial values of fields in MQXQH (continued)

Field name	Name of constant	Value of constant
Notes:		
1. The symbol 'b' represents a single blank character.		
2. The value 'Null string or blanks' denotes the null string in C, and blank characters in other programming languages.		
3. In the C programming language, the macro variable MQXQH_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:		
MQXQH MyXQH = {MQXQH_DEFAULT};		

C declaration

```
typedef struct tagMQXQH MQXQH;
struct tagMQXQH {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQCHAR48  RemoteQName;      /* Name of destination queue */
    MQCHAR48  RemoteQMGrName;   /* Name of destination queue manager */
    MQMD1     MsgDesc;          /* Original message descriptor */
};
```

COBOL declaration

```
** MQXQH structure
10 MQXQH.
** Structure identifier
15 MQXQH-STRUCID PIC X(4).
** Structure version number
15 MQXQH-VERSION PIC S9(9) BINARY.
** Name of destination queue
15 MQXQH-REMOTEQNAME PIC X(48).
** Name of destination queue manager
15 MQXQH-REMOTEQMGRNAME PIC X(48).
** Original message descriptor
15 MQXQH-MSGDESC.
** Structure identifier
20 MQXQH-MSGDESC-STRUCID PIC X(4).
** Structure version number
20 MQXQH-MSGDESC-VERSION PIC S9(9) BINARY.
** Report options
20 MQXQH-MSGDESC-REPORT PIC S9(9) BINARY.
** Message type
20 MQXQH-MSGDESC-MSGTYPE PIC S9(9) BINARY.
** Expiry time
20 MQXQH-MSGDESC-EXPIRY PIC S9(9) BINARY.
** Feedback or reason code
20 MQXQH-MSGDESC-FEEDBACK PIC S9(9) BINARY.
** Numeric encoding of message data
20 MQXQH-MSGDESC-ENCODING PIC S9(9) BINARY.
** Character set identifier of message data
20 MQXQH-MSGDESC-CODEDCHARSETID PIC S9(9) BINARY.
** Format name of message data
20 MQXQH-MSGDESC-FORMAT PIC X(8).
** Message priority
20 MQXQH-MSGDESC-PRIORITY PIC S9(9) BINARY.
** Message persistence
20 MQXQH-MSGDESC-PERSISTENCE PIC S9(9) BINARY.
** Message identifier
20 MQXQH-MSGDESC-MSGID PIC X(24).
```

MQXQH – Language declarations

```
** Correlation identifier
20 MQXQH-MSGDESC-CORRELID PIC X(24).
** Backout counter
20 MQXQH-MSGDESC-BACKOUTCOUNT PIC S9(9) BINARY.
** Name of reply-to queue
20 MQXQH-MSGDESC-REPLYTOQ PIC X(48).
** Name of reply queue manager
20 MQXQH-MSGDESC-REPLYTOQMGR PIC X(48).
** User identifier
20 MQXQH-MSGDESC-USERIDENTIFIER PIC X(12).
** Accounting token
20 MQXQH-MSGDESC-ACCOUNTINGTOKEN PIC X(32).
** Application data relating to identity
20 MQXQH-MSGDESC-APPLIDENTITYDATA PIC X(32).
** Type of application that put the message
20 MQXQH-MSGDESC-PUTAPPLTYPE PIC S9(9) BINARY.
** Name of application that put the message
20 MQXQH-MSGDESC-PUTAPPLNAME PIC X(28).
** Date when message was put
20 MQXQH-MSGDESC-PUTDATE PIC X(8).
** Time when message was put
20 MQXQH-MSGDESC-PUTTIME PIC X(8).
** Application data relating to origin
20 MQXQH-MSGDESC-APPLORIGINDATA PIC X(4).
```

PL/I declaration

```
dc1
1 MQXQH based,
3 StrucId char(4), /* Structure identifier */
3 Version fixed bin(31), /* Structure version number */
3 RemoteQName char(48), /* Name of destination queue */
3 RemoteQMgrName char(48), /* Name of destination queue
manager */
3 MsgDesc, /* Original message descriptor */
5 StrucId char(4), /* Structure identifier */
5 Version fixed bin(31), /* Structure version number */
5 Report fixed bin(31), /* Report options */
5 MsgType fixed bin(31), /* Message type */
5 Expiry fixed bin(31), /* Expiry time */
5 Feedback fixed bin(31), /* Feedback or reason code */
5 Encoding fixed bin(31), /* Numeric encoding of message
data */
5 CodedCharSetId fixed bin(31), /* Character set identifier of
message data */
5 Format char(8), /* Format name of message data */
5 Priority fixed bin(31), /* Message priority */
5 Persistence fixed bin(31), /* Message persistence */
5 MsgId char(24), /* Message identifier */
5 CorrelId char(24), /* Correlation identifier */
5 BackoutCount fixed bin(31), /* Backout counter */
5 ReplyToQ char(48), /* Name of reply-to queue */
5 ReplyToQMgr char(48), /* Name of reply queue manager */
5 UserIdentifier char(12), /* User identifier */
5 AccountingToken char(32), /* Accounting token */
5 ApplIdentityData char(32), /* Application data relating to
identity */
5 PutApplType fixed bin(31), /* Type of application that put the
message */
5 PutApplName char(28), /* Name of application that put the
message */
5 PutDate char(8), /* Date when message was put */
5 PutTime char(8), /* Time when message was put */
5 ApplOriginData char(4); /* Application data relating to
origin */
```

MQXQH – Language declarations

System/390 assembler declaration

MQXQH	DSECT	
MQXQH_STRUCID	DS	CL4 Structure identifier
MQXQH_VERSION	DS	F Structure version number
MQXQH_REMOTEQNAME	DS	CL48 Name of destination queue
MQXQH_REMOTEQMGRNAME	DS	CL48 Name of destination queue manager
*		
MQXQH_MSGDESC	DS	0F Force fullword alignment
MQXQH_MSGDESC_STRUCID	DS	CL4 Structure identifier
MQXQH_MSGDESC_VERSION	DS	F Structure version number
MQXQH_MSGDESC_REPORT	DS	F Report options
MQXQH_MSGDESC_MSGTYPE	DS	F Message type
MQXQH_MSGDESC_EXPIRY	DS	F Expiry time
MQXQH_MSGDESC_FEEDBACK	DS	F Feedback or reason code
MQXQH_MSGDESC_ENCODING	DS	F Numeric encoding of message data
*		
MQXQH_MSGDESC_CODEDCHARSETID	DS	F Character set identifier of message data
*		
MQXQH_MSGDESC_FORMAT	DS	CL8 Format name of message data
MQXQH_MSGDESC_PRIORITY	DS	F Message priority
MQXQH_MSGDESC_PERSISTENCE	DS	F Message persistence
MQXQH_MSGDESC_MSGID	DS	XL24 Message identifier
MQXQH_MSGDESC_CORRELID	DS	XL24 Correlation identifier
MQXQH_MSGDESC_BACKOUTCOUNT	DS	F Backout counter
MQXQH_MSGDESC_REPLYTOQ	DS	CL48 Name of reply-to queue
MQXQH_MSGDESC_REPLYTOQMGR	DS	CL48 Name of reply queue manager
MQXQH_MSGDESC_USERIDENTIFIER	DS	CL12 User identifier
MQXQH_MSGDESC_ACCOUNTINGTOKEN	DS	XL32 Accounting token
MQXQH_MSGDESC_APPLIDENTITYDATA	DS	CL32 Application data relating to identity
*		
MQXQH_MSGDESC_PUTAPPLTYPE	DS	F Type of application that put the message
*		
MQXQH_MSGDESC_PUTAPPLNAME	DS	CL28 Name of application that put the message
*		
MQXQH_MSGDESC_PUTDATE	DS	CL8 Date when message was put
MQXQH_MSGDESC_PUTTIME	DS	CL8 Time when message was put
MQXQH_MSGDESC_APPLORIGINDATA	DS	CL4 Application data relating to origin
*		
MQXQH_MSGDESC_LENGTH	EQU	*-MQXQH_MSGDESC
	ORG	MQXQH_MSGDESC
MQXQH_MSGDESC_AREA	DS	CL(MQXQH_MSGDESC_LENGTH)
*		
MQXQH_LENGTH	EQU	*-MQXQH
	ORG	MQXQH
MQXQH_AREA	DS	CL(MQXQH_LENGTH)

TAL declaration

```
STRUCT      MQXQH^DEF (*);
BEGIN
STRUCT      STRUCID;
BEGIN STRING BYTE [0:3]; END;
INT(32)     VERSION;
STRUCT      REMOTEQNAME;
BEGIN STRING BYTE [0:47]; END;
STRUCT      REMOTEQMGRNAME;
BEGIN STRING BYTE [0:47]; END;
STRUCT      MSGDESC(MQMD^DEF);
END;
```

Visual Basic declaration

```
Type MQXQH
  StrucId      As String*4 'Structure identifier'
  Version      As Long     'Structure version number'
```

MQXQH – Language declarations

```
RemoteQName    As String*48 'Name of destination queue'  
RemoteQMgrName As String*48 'Name of destination queue manager'  
MsgDesc       As MQMD1     'Original message descriptor'  
End Type
```

MQXQH – Language declarations

Part 2. Function calls

Chapter 26. Call descriptions	331	CompCode (MQLONG) – output	353
Conventions used in the call descriptions	331	Reason (MQLONG) – output	354
Using the calls in the C language.	333	Usage notes	354
Declaring the Buffer parameter	333	Language invocations	356
		C invocation.	356
		COBOL invocation	357
		PL/I invocation	357
		System/390 assembler invocation.	357
		TAL invocation	357
		Visual Basic invocation	357
Chapter 27. MQBACK – Back out changes.	335		
Syntax.	335	Chapter 31. MQCONN – Connect queue manager	359
Parameters	335	Syntax.	359
Hconn (MQHCONN) – input	335	Parameters	359
CompCode (MQLONG) – output.	335	QMgrName (MQCHAR48) – input	359
Reason (MQLONG) – output	335	Hconn (MQHCONN) – output	361
Usage notes	336	CompCode (MQLONG) – output.	362
Language invocations	338	Reason (MQLONG) – output	363
C invocation.	338	Usage notes	364
COBOL invocation	339	Language invocations	366
PL/I invocation	339	C invocation.	366
System/390 assembler invocation.	339	COBOL invocation	366
TAL invocation	339	PL/I invocation	366
Visual Basic invocation	339	System/390 assembler invocation.	367
		TAL invocation	367
		Visual Basic invocation	367
Chapter 28. MQBEGIN – Begin unit of work	341		
Syntax.	341	Chapter 32. MQCONNX – Connect queue manager (extended)	369
Parameters	341	Syntax.	369
Hconn (MQHCONN) – input	341	Parameters	369
BeginOptions (MQBO) – input/output	341	QMgrName (MQCHAR48) – input	369
CompCode (MQLONG) – output.	341	ConnectOpts (MQCNO) – input/output	369
Reason (MQLONG) – output	342	Hconn (MQHCONN) – output	369
Usage notes	342	CompCode (MQLONG) – output.	369
Language invocations	344	Reason (MQLONG) – output	369
C invocation.	344	Usage notes	371
COBOL invocation	344	Language invocations	371
PL/I invocation	344	C invocation.	371
Visual Basic invocation	344	COBOL invocation	371
		PL/I invocation	371
		System/390 assembler invocation.	372
		Visual Basic invocation	372
Chapter 29. MQCLOSE – Close object	345		
Syntax.	345	Chapter 33. MQDISC – Disconnect queue manager	373
Parameters	345	Syntax.	373
Hconn (MQHCONN) – input	345	Parameters	373
Hobj (MQHOBJ) – input/output	345	Hconn (MQHCONN) – input/output	373
Options (MQLONG) – input	345	CompCode (MQLONG) – output.	373
CompCode (MQLONG) – output.	347	Reason (MQLONG) – output	374
Reason (MQLONG) – output	347	Usage notes	375
Usage notes	348	Language invocations	376
Language invocations	350	C invocation.	376
C invocation.	350		
COBOL invocation	351		
PL/I invocation	351		
System/390 assembler invocation.	351		
TAL invocation	351		
Visual Basic invocation	351		
Chapter 30. MQCMIT – Commit changes	353		
Syntax.	353		
Parameters	353		
Hconn (MQHCONN) – input	353		

Function calls

COBOL invocation	376	C invocation.	421
PL/I invocation	376	COBOL invocation	421
System/390 assembler invocation.	376	PL/I invocation	421
TAL invocation	376	System/390 assembler invocation.	421
Visual Basic invocation	377	TAL invocation	422
		Visual Basic invocation	422
Chapter 34. MQGET – Get message	379	Chapter 37. MQPUT – Put message	423
Syntax.	379	Syntax.	423
Parameters	379	Parameters	423
Hconn (MQHCONN) – input	379	Hconn (MQHCONN) – input	423
Hobj (MQHOBJ) – input.	379	Hobj (MQHOBJ) – input.	423
MsgDesc (MQMD) – input/output	379	MsgDesc (MQMD) – input/output	423
GetMsgOpts (MQGMO) – input/output	380	PutMsgOpts (MQPMO) – input/output	424
BufferLength (MQLONG) – input	380	BufferLength (MQLONG) – input	424
Buffer (MQBYTE×BufferLength) – output	380	Buffer (MQBYTE×BufferLength) – input	425
DataLength (MQLONG) – output	381	CompCode (MQLONG) – output.	425
CompCode (MQLONG) – output.	381	Reason (MQLONG) – output	425
Reason (MQLONG) – output	381	Usage notes	430
Usage notes	385	Language invocations	434
Language invocations	389	C invocation.	434
C invocation.	389	COBOL invocation	434
COBOL invocation	389	PL/I invocation	435
PL/I invocation	390	System/390 assembler invocation.	435
System/390 assembler invocation.	390	TAL invocation.	435
TAL invocation.	390	Visual Basic invocation	435
Visual Basic invocation	391		
Chapter 35. MQINQ – Inquire object attributes	393	Chapter 38. MQPUT1 – Put one message	437
Syntax.	393	Syntax.	437
Parameters	393	Parameters	437
Hconn (MQHCONN) – input	393	Hconn (MQHCONN) – input	437
Hobj (MQHOBJ) – input.	393	ObjDesc (MQOD) – input/output	437
SelectorCount (MQLONG) – input	393	MsgDesc (MQMD) – input/output	437
Selectors (MQLONG×SelectorCount) – input	394	PutMsgOpts (MQPMO) – input/output	438
IntAttrCount (MQLONG) – input	397	BufferLength (MQLONG) – input	438
IntAttrs (MQLONG×IntAttrCount) – output	397	Buffer (MQBYTE×BufferLength) – input	438
CharAttrLength (MQLONG) – input	398	CompCode (MQLONG) – output.	438
CharAttrs (MQCHAR×CharAttrLength) – output.	398	Reason (MQLONG) – output	438
CompCode (MQLONG) – output.	398	Usage notes	443
Reason (MQLONG) – output	398	Language invocations	445
Usage notes	400	C invocation.	445
Language invocations	401	COBOL invocation	445
C invocation.	401	PL/I invocation	445
COBOL invocation	401	System/390 assembler invocation.	446
PL/I invocation	402	TAL invocation.	446
System/390 assembler invocation.	402	Visual Basic invocation	446
TAL invocation.	402		
Visual Basic invocation	403	Chapter 39. MQSET – Set object attributes	447
Chapter 36. MQOPEN – Open object	405	Syntax.	447
Syntax.	405	Parameters	447
Parameters	405	Hconn (MQHCONN) – input	447
Hconn (MQHCONN) – input	405	Hobj (MQHOBJ) – input.	447
ObjDesc (MQOD) – input/output	405	SelectorCount (MQLONG) – input	447
Options (MQLONG) – input	406	Selectors (MQLONG×SelectorCount) – input	447
Hobj (MQHOBJ) – output	412	IntAttrCount (MQLONG) – input	448
CompCode (MQLONG) – output.	412	IntAttrs (MQLONG×IntAttrCount) – input	448
Reason (MQLONG) – output	412	CharAttrLength (MQLONG) – input	449
Usage notes	415	CharAttrs (MQCHAR×CharAttrLength) – input	449
Language invocations	421	CompCode (MQLONG) – output.	449
		Reason (MQLONG) – output	449
		Usage notes	451

Language invocations	452
C invocation.	452
COBOL invocation	452
PL/I invocation	452
System/390 assembler invocation.	453
TAL invocation.	453
Visual Basic invocation	453

Function calls

Chapter 26. Call descriptions

This part of the book describes the MQI calls:

- MQBACK – Back out
- MQBEGIN – Begin unit of work
- MQCLOSE – Close object
- MQCMIT – Commit
- MQCONN – Connect to queue manager
- MQCONNX – Connect to queue manager with options
- MQDISC – Disconnect from queue manager
- MQGET – Get message
- MQINQ – Inquire about object attributes
- MQOPEN – Open object
- MQPUT – Put message
- MQPUT1 – Put one message
- MQSET – Set object attributes
- MQSYNC – Synchronize statistics updates (Compaq NSK only)

Online help on the UNIX platforms, in the form of *man* pages, is available for these calls.

Note: The calls associated with data conversion, MQXCNVC and MQ_DATA_CONV_EXIT, are in Appendix F, “Data conversion”, on page 581.

Conventions used in the call descriptions

For each call, this chapter gives a description of the parameters and usage of the call in a format that is independent of programming language. This is followed by typical invocations of the call, and typical declarations of its parameters, in each of the supported programming languages.

The description of each call contains the following sections:

Call name

The call name, followed by a brief description of the purpose of the call.

Parameters

For each parameter, the name is followed by its data type in parentheses () and one of the following:

input You supply information in the parameter when you make the call.

output

The queue manager returns information in the parameter when the call completes or fails.

input/output

You supply information in the parameter when you make the call, and the queue manager changes the information when the call completes or fails.

For example:

Compcode (MQLONG) — output

Call descriptions

In some cases, the data type is a structure. In all cases, there is more information about the data type or structure in “Elementary data types” on page 7.

The last two parameters in each call are a completion code and a reason code. The completion code indicates whether the call completed successfully, partially, or not at all. Further information about the partial success or the failure of the call is given in the reason code. You will find more information about each completion and reason code in Appendix A, “Return codes”, on page 527.

Usage notes

Additional information about the call, describing how to use it and any restrictions on its use.

Assembler language invocation

Typical invocation of the call, and declaration of its parameters, in assembler language.

C invocation

Typical invocation of the call, and declaration of its parameters, in C.

COBOL invocation

Typical invocation of the call, and declaration of its parameters, in COBOL.

PL/I invocation

Typical invocation of the call, and declaration of its parameters, in PL/I.

All parameters are passed by reference.

TAL invocation

Typical invocation of the call, and declaration of its parameters, in TAL.

Visual Basic invocation

Typical invocation of the call, and declaration of its parameters, in Visual Basic.

Other notation conventions are:

Constants

Names of constants are shown in uppercase; for example, MQOO_OUTPUT. A set of constants having the same prefix is shown like this: MQIA_*. See Appendix B, “MQ constants”, on page 529 for the value of a constant.

Arrays

In some calls, parameters are arrays of character strings whose size is not fixed. In the descriptions of these parameters, a lowercase “n” represents a numeric constant. When you code the declaration for that parameter, replace the “n” with the numeric value you require.

Using the calls in the C language

Parameters that are *input only* and of type MQHCONN, MQHOBJ, or MQLONG are passed by value. For all other parameters, the *address* of the parameter is passed by value.

Not all parameters that are passed by address need to be specified every time a function is invoked. Where a particular parameter is not required, a null pointer can be specified as the parameter on the function invocation, in place of the address of parameter data. Parameters for which this is possible are identified in the call descriptions.

No parameter is returned as the value of the call; in C terminology, this means that all calls return **void**.

Declaring the Buffer parameter

The MQGET, MQPUT, and MQPUT1 calls each have one parameter that has an undefined data type—the *Buffer* parameter. This parameter is used to send and receive the application's message data.

Parameters of this sort are shown in the C examples as arrays of MQBYTE. It is perfectly valid to declare the parameters in this way, but it is usually more convenient to declare them as the particular structure that describes the layout of the data in the message. The function prototype declares the parameter as a pointer-to-void, so that you can specify the address of any sort of data as the parameter on the call invocation.

Pointer-to-void is a pointer to data of undefined format. It is defined as:

```
typedef void *PMQVOID;
```

Call descriptions

Chapter 27. MQBACK – Back out changes

The MQBACK call indicates to the queue manager that all the message gets and puts that have occurred since the last syncpoint are to be backed out. Messages put as part of a unit of work are deleted; messages retrieved as part of a unit of work are reinstated on the queue.

- On z/OS, this call is used only by batch programs (including IMS batch DL/I programs). On WebSphere Application Server embedded messaging using reduced function WebSphere MQ, this call can be used only by batch programs (**not** including IMS batch DL/I programs) when the unit of work affects only MQ resources.
- On OS/400, this call is not supported for applications running in compatibility mode.
- On Compaq NonStop Kernel, this call can be issued by the application but always returns completion code MQCC_FAILED and reason code MQRC_ENVIRONMENT_ERROR.

MQBACK is valid for client applications to Compaq NonStop Kernel, and **must** be used to back out messages that have been put or got using MQPMO_SYNCPOINT or MQGMO_SYNCPOINT.

- On VSE/ESA, this call is used only by client programs and batch programs.

Syntax

MQBACK (*Hconn*, *CompCode*, *Reason*)

Parameters

The MQBACK call has the following parameters.

Hconn (MQHCONN) – input

Connection handle.

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQBACK – Reason parameter

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_SERV_LOAD_ERROR

(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR

(2374, X'946') API exit failed.

MQRC_ASID_MISMATCH

(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS

(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_STRUC_IN_USE

(2346, X'92A') Coupling-facility structure in use.

MQRC_CONNECTION_BROKEN

(2009, X'7D9') Connection to queue manager lost.

MQRC_ENVIRONMENT_ERROR

(2012, X'7DC') Call not valid in environment.

MQRC_HCONN_ERROR

(2018, X'7E2') Connection handle not valid.

MQRC_OBJECT_DAMAGED

(2101, X'835') Object damaged.

MQRC_OUTCOME_MIXED

(2123, X'84B') Result of commit or back-out operation is mixed.

MQRC_Q_MGR_STOPPING

(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM

(2102, X'836') Insufficient system resources available.

MQRC_STORAGE_MEDIUM_FULL

(2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE

(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR

(2195, X'893') Unexpected error occurred.

See Appendix A, "Return codes", on page 527 for more details.

Usage notes

1. This call can be used only when the queue manager itself coordinates the unit of work. This can be:
 - A local unit of work, where the changes affect only MQ resources.
 - A global unit of work, where the changes can affect resources belonging to other resource managers, as well as affecting MQ resources.

For further details about local and global units of work, see Chapter 28, "MQBEGIN – Begin unit of work", on page 341.

2. In environments where the queue manager does not coordinate the unit of work, the appropriate back-out call must be used instead of MQBACK. The environment may also support an implicit back out caused by the application terminating abnormally.
 - On z/OS, the following calls should be used:
 - Batch programs (including IMS batch DL/I programs) can use the MQBACK call if the unit of work affects only MQ resources. However, if the unit of work affects both MQ resources and resources belonging to

other resource managers (for example, DB2[®]), the SRRBACK call provided by the z/OS Recoverable Resource Service (RRS) should be used. The SRRBACK call backs out changes to resources belonging to the resource managers that have been enabled for RRS coordination.

(On WebSphere Application Server embedded messaging using reduced function WebSphere MQ, this call can be used only by batch programs (**not** including IMS batch DL/I programs) when the unit of work affects only MQ resources.)

- CICS applications should use the EXEC CICS SYNCPOINT ROLLBACK command to back out the unit of work. The MQBACK call cannot be used for CICS applications.
 - IMS applications (other than batch DL/I programs) should use IMS calls such as ROLB to back out the unit of work. The MQBACK call cannot be used for IMS applications (other than batch DL/I programs).
 - On OS/400, this call can be used for local units of work coordinated by the queue manager. This means that a commitment definition must not exist at job level, that is, the STRCMTCTL command with the CMTSCOPE(*JOB) parameter must not have been issued for the job.
 - On Compaq NonStop Kernel, this call always returns a *CompCode* of MQCC_FAILED and a *Reason* of MQRC_ENVIRONMENT_ERROR. Transactions are managed externally through TM/MP.
 - On VSE/ESA, this call is used only by client programs and batch programs. In both cases the call causes the queue manager to issue the EXEC CICS SYNCPOINT ROLLBACK command on behalf of the application.
- This call is not supported for CICS applications, which should use instead the EXEC CICS SYNCPOINT ROLLBACK command to cause changes to be backed out. Changes are also backed out if the application terminates abnormally; in this situation CICS executes a dynamic transaction backout (DTB) on behalf of the application.
3. If an application ends with uncommitted changes in a unit of work, the disposition of those changes depends on whether the application ends normally or abnormally. See the usage notes in Chapter 33, “MQDISC – Disconnect queue manager”, on page 373 for further details.
 4. When an application puts or gets messages in groups or segments of logical messages, the queue manager retains information relating to the message group and logical message for the last successful MQPUT and MQGET calls. This information is associated with the queue handle, and includes such things as:
 - The values of the *GroupId*, *MsgSeqNumber*, *Offset*, and *MsgFlags* fields in MQMD.
 - Whether the message is part of a unit of work.
 - For the MQPUT call: whether the message is persistent or nonpersistent.

The queue manager keeps *three* sets of group and segment information, one set for each of the following:

- The last successful MQPUT call (this can be part of a unit of work).
- The last successful MQGET call that removed a message from the queue (this can be part of a unit of work).
- The last successful MQGET call that browsed a message on the queue (this *cannot* be part of a unit of work).

If the application puts or gets the messages as part of a unit of work, and the application then decides to back out the unit of work, the group and segment information is restored to the value that it had previously:

MQBACK – Usage notes

- The information associated with the MQPUT call is restored to the value that it had prior to the first successful MQPUT call for that queue handle in the current unit of work.
- The information associated with the MQGET call is restored to the value that it had prior to the first successful MQGET call for that queue handle in the current unit of work.

Queues which were updated by the application after the unit of work had started, but outside the scope of the unit of work, do not have their group and segment information restored if the unit of work is backed out.

Restoring the group and segment information to its previous value when a unit of work is backed out allows the application to spread a large message group or large logical message consisting of many segments across several units of work, and to restart at the correct point in the message group or logical message if one of the units of work fails. Using several units of work may be advantageous if the local queue manager has only limited queue storage. However, the application must maintain sufficient information to be able to restart putting or getting messages at the correct point in the event that a system failure occurs. For details of how to restart at the correct point after a system failure, see the MQPMO_LOGICAL_ORDER option described in Chapter 14, “MQPMO – Put-message options”, on page 229, and the MQGMO_LOGICAL_ORDER option described in Chapter 8, “MQGMO – Get-message options”, on page 95.

The remaining usage notes apply only when the queue manager coordinates the units of work:

5. A unit of work has the same scope as a connection handle. This means that all MQ calls which affect a particular unit of work must be performed using the same connection handle. Calls issued using a different connection handle (for example, calls issued by another application) affect a different unit of work. See the *Hconn* parameter described in Chapter 31, “MQCONN – Connect queue manager”, on page 359 for information about the scope of connection handles.
6. Only messages that were put or retrieved as part of the current unit of work are affected by this call.
7. A long-running application that issues MQGET, MQPUT, or MQPUT1 calls within a unit of work, but which never issues a commit or backout call, can cause queues to fill up with messages that are not available to other applications. To guard against this possibility, the administrator should set the *MaxUncommittedMsgs* queue-manager attribute to a value that is low enough to prevent runaway applications filling the queues, but high enough to allow the expected messaging applications to work correctly.

Language invocations

The MQBACK call is supported in the programming languages shown below.

C invocation

```
MQBACK (Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;    /* Connection handle */
MQLONG   CompCode; /* Completion code */
MQLONG   Reason;   /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQBACK' USING HCONN, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN    PIC S9(9) BINARY.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON   PIC S9(9) BINARY.
```

PL/I invocation

```
call MQBACK (Hconn, CompCode, Reason);
```

Declare the parameters as follows:

```
dc1 Hconn    fixed bin(31); /* Connection handle */
dc1 CompCode fixed bin(31); /* Completion code */
dc1 Reason   fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
CALL MQBACK,(HCONN,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN    DS F Connection handle
COMPCODE DS F Completion code
REASON   DS F Reason code qualifying COMPCODE
```

TAL invocation

```
INT(32) .EXT Hconn;
INT(32) .EXT CC;
INT(32) .EXT Reason;
```

```
CALL MQBACK(HConn, CC, Reason);
```

Visual Basic invocation

```
MQBACK Hconn, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn    As Long 'Connection handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

MQBACK – Language invocations

Chapter 28. MQBEGIN – Begin unit of work

The MQBEGIN call begins a unit of work that is coordinated by the queue manager, and that may involve external resource managers.

- This call is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Solaris, Linux, Windows.
- On Compaq NonStop Kernel, this call can be issued by the application but always returns completion code MQCC_FAILED and reason code MQRC_ENVIRONMENT_ERROR.

Syntax

MQBEGIN (*Hconn*, *BeginOptions*, *CompCode*, *Reason*)

Parameters

The MQBEGIN call has the following parameters.

Hconn (MQHCONN) – input

Connection handle.

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

Hconn must be a nonshared connection handle. If a shared connection handle is specified, the call fails with reason code MQRC_HCONN_ERROR. See the description of the MQCNO_HANDLE_SHARE_* options in Chapter 5, “MQCNO – Connect options”, on page 61 for more information about shared and nonshared handles.

BeginOptions (MQBO) – input/output

Options that control the action of MQBEGIN.

See Chapter 3, “MQBO – Begin options”, on page 37 for details.

If no options are required, programs written in C or S/390 assembler can specify a null parameter address, instead of specifying the address of an MQBO structure.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_WARNING

Warning (partial completion).

MQCC_FAILED

Call failed.

MQBEGIN – Reason parameter

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_NO_EXTERNAL_PARTICIPANTS

(2121, X'849') No participating resource managers registered.

MQRC_PARTICIPANT_NOT_AVAILABLE

(2122, X'84A') Participating resource manager not available.

If *CompCode* is MQCC_FAILED:

MQRC_API_EXIT_ERROR

(2374, X'946') API exit failed.

MQRC_BO_ERROR

(2134, X'856') Begin-options structure not valid.

MQRC_CALL_IN_PROGRESS

(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CONNECTION_BROKEN

(2009, X'7D9') Connection to queue manager lost.

MQRC_ENVIRONMENT_ERROR

(2012, X'7DC') Call not valid in environment.

MQRC_HCONN_ERROR

(2018, X'7E2') Connection handle not valid.

MQRC_OPTIONS_ERROR

(2046, X'7FE') Options not valid or not consistent.

MQRC_Q_MGR_STOPPING

(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM

(2102, X'836') Insufficient system resources available.

MQRC_STORAGE_NOT_AVAILABLE

(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR

(2195, X'893') Unexpected error occurred.

MQRC_UOW_IN_PROGRESS

(2128, X'850') Unit of work already started.

For more information on these reason codes, see Appendix A, "Return codes", on page 527.

Usage notes

1. The MQBEGIN call can be used to start a unit of work that is coordinated by the queue manager and that may involve changes to resources owned by other resource managers. The queue manager supports three types of unit-of-work:
 - **Queue-manager-coordinated local unit of work:** This is a unit of work in which the queue manager is the only resource manager participating, and so the queue manager acts as the unit-of-work coordinator.
 - To start this type of unit of work, the MQPMO_SYNCPOINT or MQGMO_SYNCPOINT option should be specified on the first MQPUT, MQPUT1, or MQGET call in the unit of work.

It is not necessary for the application to issue the MQBEGIN call to start the unit of work, but if MQBEGIN is used, the call completes with

MQCC_WARNING and reason code
MQRC_NO_EXTERNAL_PARTICIPANTS.

- To commit or back out this type of unit of work, the MQCMIT or MQBACK call must be used.
 - **Queue-manager-coordinated global unit of work:** This is a unit of work in which the queue manager acts as the unit-of-work coordinator, both for MQ resources *and* for resources belonging to other resource managers. Those resource managers cooperate with the queue manager to ensure that all changes to resources in the unit of work are committed or backed out together.
 - To start this type of unit of work, the MQBEGIN call must be used.
 - To commit or back out this type of unit of work, the MQCMIT and MQBACK calls must be used.
 - **Externally-coordinated global unit of work:** This is a unit of work in which the queue manager is a participant, but the queue manager does not act as the unit-of-work coordinator. Instead, there is an external unit-of-work coordinator with whom the queue manager cooperates.
 - To start this type of unit of work, the relevant call provided by the external unit-of-work coordinator must be used.
If the MQBEGIN call is used to try to start the unit of work, the call fails with reason code MQRC_ENVIRONMENT_ERROR.
 - To commit or back out this type of unit of work, the commit and back-out calls provided by the external unit-of-work coordinator must be used.
If the MQCMIT or MQBACK call is used to try to commit or back out the unit of work, the call fails with reason code MQRC_ENVIRONMENT_ERROR.
2. If the application ends with uncommitted changes in a unit of work, the disposition of those changes depends on whether the application ends normally or abnormally. See the usage notes in Chapter 33, “MQDISC – Disconnect queue manager”, on page 373 for further details.
 3. An application can participate in only one unit of work at a time. The MQBEGIN call fails with reason code MQRC_UOW_IN_PROGRESS if there is already a unit of work in existence for the application, regardless of which type of unit of work it is.
 4. The MQBEGIN call is not valid in an MQ client environment. An attempt to use the call fails with reason code MQRC_ENVIRONMENT_ERROR.
 5. When the queue manager is acting as the unit-of-work coordinator for global units of work, the resource managers that can participate in the unit of work are defined in the queue manager’s configuration file.
 6. On OS/400, the three types of unit of work are supported as follows:
 - **Queue-manager-coordinated local unit of work** can be used only when a commitment definition does not exist at the job level, that is, the STRCMTCTL command with the CMTSCOPE(*JOB) parameter must not have been issued for the job.
 - **Queue-manager-coordinated global unit of work** is not supported.
 - **Externally-coordinated global unit of work** can be used only when a commitment definition exists at job level, that is, the STRCMTCTL command with the CMTSCOPE(*JOB) parameter must have been issued for the job. If this has been done, the OS/400 COMMIT and ROLLBACK operations apply to MQ resources as well as to resources belonging to other participating resource managers.

Language invocations

The MQBEGIN call is supported in the programming languages shown below.

C invocation

```
MQBEGIN (Hconn, &BeginOptions, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;          /* Connection handle */
MQBO     BeginOptions;   /* Options that control the action of MQBEGIN */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQBEGIN' USING HCONN, BEGINOPTIONS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN          PIC S9(9) BINARY.
** Options that control the action of MQBEGIN
01 BEGINOPTIONS.
   COPY CMQBOV.
** Completion code
01 COMPCODE       PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON         PIC S9(9) BINARY.
```

PL/I invocation

```
call MQBEGIN (Hconn, BeginOptions, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn          fixed bin(31); /* Connection handle */
dcl BeginOptions  like MQBO;     /* Options that control the action of
                                MQBEGIN */
dcl CompCode      fixed bin(31); /* Completion code */
dcl Reason        fixed bin(31); /* Reason code qualifying CompCode */
```

Visual Basic invocation

```
MQBEGIN Hconn, BeginOptions, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn          As Long 'Connection handle'
Dim BeginOptions  As MQBO 'Options that control the action of MQBEGIN'
Dim CompCode      As Long 'Completion code'
Dim Reason        As Long 'Reason code qualifying CompCode'
```

Chapter 29. MQCLOSE – Close object

The MQCLOSE call relinquishes access to an object, and is the inverse of the MQOPEN call.

Syntax

MQCLOSE (*Hconn*, *Hobj*, *Options*, *CompCode*, *Reason*)

Parameters

The MQCLOSE call has the following parameters.

Hconn (MQHCONN) – input

Connection handle.

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on OS/400 for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Hobj (MQHOBJ) – input/output

Object handle.

This handle represents the object that is being closed. The object can be of any type. The value of *Hobj* was returned by a previous MQOPEN call.

On successful completion of the call, the queue manager sets this parameter to a value that is not a valid handle for the environment. This value is:

MQHO_UNUSABLE_HOBJ
Unusable object handle.

On z/OS, *Hobj* is set to a value that is undefined.

Options (MQLONG) – input

Options that control the action of MQCLOSE.

The *Options* parameter controls how the object is closed. Only permanent dynamic queues can be closed in more than one way, being either retained or deleted; these are queues whose *DefinitionType* attribute has the value MQQDT_PERMANENT_DYNAMIC (see the *DefinitionType* attribute described in Chapter 40, “Attributes for queues”, on page 457). The close options are summarized in Table 84 on page 347.

MQCLOSE – Options parameter

One (and only one) of the following must be specified:

MQCO_NONE

No optional close processing required.

This *must* be specified for:

- Objects other than queues
- Predefined queues
- Temporary dynamic queues (but only in those cases where *Hobj* is *not* the handle returned by the MQOPEN call that created the queue).
- Distribution lists

In all of the above cases, the object is retained and not deleted.

If this option is specified for a temporary dynamic queue:

- The queue is deleted, if it was created by the MQOPEN call that returned *Hobj*; any messages that are on the queue are purged.
- In all other cases the queue (and any messages on it) are retained.

If this option is specified for a permanent dynamic queue, the queue is retained and not deleted.

On z/OS, if the queue is a dynamic queue that has been logically deleted, and this is the last handle for it, the queue is physically deleted. See the usage notes for further details.

MQCO_DELETE

Delete the queue.

The queue is deleted if either of the following is true:

- It is a permanent dynamic queue, and there are no messages on the queue and no uncommitted get or put requests outstanding for the queue (either for the current task or any other task).
- It is the temporary dynamic queue that was created by the MQOPEN call that returned *Hobj*. In this case, all the messages on the queue are purged.

In all other cases the call fails with reason code MQRC_OPTION_NOT_VALID_FOR_TYPE, and the object is not deleted.

On z/OS, if the queue is a dynamic queue that has been logically deleted, and this is the last handle for it, the queue is physically deleted. See the usage notes for further details.

MQCO_DELETE_PURGE

Delete the queue, purging any messages on it.

The queue is deleted if either of the following is true:

- It is a permanent dynamic queue and there are no uncommitted get or put requests outstanding for the queue (either for the current task or any other task).
- It is the temporary dynamic queue that was created by the MQOPEN call that returned *Hobj*.

In all other cases the call fails with reason code MQRC_OPTION_NOT_VALID_FOR_TYPE, and the object is not deleted.

MQCLOSE – Options parameter

Table 84. Effect of MQCLOSE options on various types of object and queue. This table shows which close options are valid, and whether the object is retained or deleted.

Type of object or queue	MQCO_NONE	MQCO_DELETE	MQCO_DELETE_PURGE
Object other than a queue	Retained	Not valid	Not valid
Predefined queue	Retained	Not valid	Not valid
Permanent dynamic queue	Retained	Deleted if empty and no pending updates	Messages deleted; queue deleted if no pending updates
Temporary dynamic queue (call issued by creator of queue)	Deleted	Deleted	Deleted
Temporary dynamic queue (call not issued by creator of queue)	Retained	Not valid	Not valid
Distribution list	Retained	Not valid	Not valid

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_WARNING

Warning (partial completion).

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_INCOMPLETE_GROUP

(2241, X'8C1') Message group not complete.

MQRC_INCOMPLETE_MSG

(2242, X'8C2') Logical message not complete.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE

(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR

(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR

(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR

(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH

(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS

(2219, X'8AB') MQI call entered before previous call complete.

MQCLOSE – Reason parameter

	MQRC_CF_STRUC_FAILED (2373, X'945')	Coupling-facility structure failed.
	MQRC_CF_STRUC_IN_USE (2346, X'92A')	Coupling-facility structure in use.
	MQRC_CICS_WAIT_FAILED (2140, X'85C')	Wait request rejected by CICS.
	MQRC_CONNECTION_BROKEN (2009, X'7D9')	Connection to queue manager lost.
	MQRC_CONNECTION_NOT_AUTHORIZED (2217, X'8A9')	Not authorized for connection.
	MQRC_CONNECTION_STOPPING (2203, X'89B')	Connection shutting down.
	MQRC_HCONN_ERROR (2018, X'7E2')	Connection handle not valid.
	MQRC_HOBJ_ERROR (2019, X'7E3')	Object handle not valid.
	MQRC_NOT_AUTHORIZED (2035, X'7F3')	Not authorized for access.
	MQRC_OBJECT_DAMAGED (2101, X'835')	Object damaged.
	MQRC_OPTION_NOT_VALID_FOR_TYPE (2045, X'7FD')	Option not valid for object type.
	MQRC_OPTIONS_ERROR (2046, X'7FE')	Options not valid or not consistent.
	MQRC_PAGESET_ERROR (2193, X'891')	Error accessing page-set data set.
	MQRC_Q_MGR_NAME_ERROR (2058, X'80A')	Queue manager name not valid or not known.
	MQRC_Q_MGR_NOT_AVAILABLE (2059, X'80B')	Queue manager not available for connection.
	MQRC_Q_MGR_STOPPING (2162, X'872')	Queue manager shutting down.
	MQRC_Q_NOT_EMPTY (2055, X'807')	Queue contains one or more messages or uncommitted put or get requests.
	MQRC_RESOURCE_PROBLEM (2102, X'836')	Insufficient system resources available.
	MQRC_SECURITY_ERROR (2063, X'80F')	Security error occurred.
	MQRC_STORAGE_NOT_AVAILABLE (2071, X'817')	Insufficient storage available.
	MQRC_SUPPRESSED_BY_EXIT (2109, X'83D')	Call suppressed by exit program.
	MQRC_UNEXPECTED_ERROR (2195, X'893')	Unexpected error occurred.

See Appendix A, "Return codes", on page 527 for more details.

Usage notes

1. When an application issues the MQDISC call, or ends either normally or abnormally, any objects that were opened by the application and are still open are closed automatically with the MQCO_NONE option.
2. The following points apply if the object being closed is a *queue*:

- If operations on the queue were performed as part of a unit of work, the queue can be closed before or after the syncpoint occurs without affecting the outcome of the syncpoint.
 - If the queue was opened with the MQOO_BROWSE option, the browse cursor is destroyed. If the queue is subsequently reopened with the MQOO_BROWSE option, a new browse cursor is created (see the MQOO_BROWSE option described in MQOPEN).
 - If a message is currently locked for this handle at the time of the MQCLOSE call, the lock is released (see the MQGMO_LOCK option described in Chapter 8, “MQGMO – Get-message options”, on page 95).
 - On z/OS, if there is an MQGET request with the MQGMO_SET_SIGNAL option outstanding against the queue handle being closed, the request is canceled (see the MQGMO_SET_SIGNAL option described in Chapter 8, “MQGMO – Get-message options”, on page 95). Signal requests for the same queue but lodged against different handles (*Hobj*) are not affected (unless it is a dynamic queue that is being deleted, in which case they are also canceled).
3. The following points apply if the object being closed is a *dynamic queue* (either permanent or temporary):
- For a dynamic queue, the options MQCO_DELETE or MQCO_DELETE_PURGE can be specified regardless of the options specified on the corresponding MQOPEN call.
 - When a dynamic queue is deleted, all MQGET calls with the MQGMO_WAIT option that are outstanding against the queue are canceled and reason code MQRC_Q_DELETED is returned. See the MQGMO_WAIT option described in Chapter 8, “MQGMO – Get-message options”, on page 95.

After a dynamic queue has been deleted, any call (other than MQCLOSE) that attempts to reference the queue using a previously acquired *Hobj* handle fails with reason code MQRC_Q_DELETED.

Be aware that although a deleted queue cannot be accessed by applications, the queue is not removed from the system, and associated resources are not freed, until such time as all handles that reference the queue have been closed, and all units of work that affect the queue have been either committed or backed out.

On z/OS, a queue that has been logically deleted but not yet removed from the system prevents the creation of a new queue with the same name as the deleted queue; the MQOPEN call fails with reason code MQRC_NAME_IN_USE in this case. Also, such a queue can still be displayed using MQSC commands, even though it cannot be accessed by applications.

- When a permanent dynamic queue is deleted, if the *Hobj* handle specified on the MQCLOSE call is *not* the one that was returned by the MQOPEN call that created the queue, a check is made that the user identifier which was used to validate the MQOPEN call is authorized to delete the queue. If the MQOO_ALTERNATE_USER_AUTHORITY option was specified on the MQOPEN call, the user identifier checked is the *AlternateUserId*.

This check is not performed if:

- The handle specified is the one returned by the MQOPEN call that created the queue.
- The queue being deleted is a temporary dynamic queue.

MQCLOSE – Usage notes

- When a temporary dynamic queue is closed, if the *Hobj* handle specified on the MQCLOSE call is the one that was returned by the MQOPEN call that created the queue, the queue is deleted. This occurs regardless of the close options specified on the MQCLOSE call. If there are messages on the queue, they are discarded; no report messages are generated.

If there are uncommitted units of work that affect the queue, the queue and its messages are still deleted, but this does not cause the units of work to fail. However, as described above, the resources associated with the units of work are not freed until each of the units of work has been either committed or backed out.

4. The following points apply if the object being closed is a *distribution list*:
 - The only valid close option for a distribution list is MQCO_NONE; the call fails with reason code MQRC_OPTIONS_ERROR or MQRC_OPTION_NOT_VALID_FOR_TYPE if any other options are specified.
 - When a distribution list is closed, individual completion codes and reason codes are not returned for the queues in the list – only the *CompCode* and *Reason* parameters of the call are available for diagnostic purposes.

If a failure occurs closing one of the queues, the queue manager continues processing and attempts to close the remaining queues in the distribution list. The *CompCode* and *Reason* parameters of the call are then set to return information describing the failure. Thus it is possible for the completion code to be MQCC_FAILED, even though most of the queues were closed successfully. The queue that encountered the error is not identified.

If there is a failure on more than one queue, it is not defined which failure is reported in the *CompCode* and *Reason* parameters.

5. On OS/400, if the application was connected implicitly when the first MQOPEN call was issued, an implicit MQDISC occurs when the last MQCLOSE is issued.

Only applications running in compatibility mode can be connected implicitly; other applications must issue the MQCONN or MQCONNX call to connect to the queue manager explicitly.

6. On MQSeries for Compaq NonStop Kernel, if there is an MQGET request with the MQGMO_SET_SIGNAL option outstanding against the queue handle being closed, the request is canceled. Signal requests for the same queue, but lodged against different handles (*Hobj*), are not affected (unless it is a dynamic queue that is being deleted, in which case, they are also canceled.)

For a FASTPATH application opening or closing a dynamic queue, MQSeries can start and end a TM/MP transaction in order to update audited databases. If the application has opened the TM/MP T-file (because it can initiate multiple transactions), ENDTRANSACTION is a no-waited operation, and the application receives a completion for the transaction initiated by MQSeries. Review the design of applications to determine if this is the case and verify that the logic handling completions can cope with ENDTRANSACTION completions that are caused by MQSeries.

Language invocations

The MQCLOSE call is supported in the programming languages shown below.

C invocation

```
MQCLOSE (Hconn, &Hobj, Options, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;    /* Connection handle */
MQHOBJ  Hobj;     /* Object handle */
MQLONG  Options;  /* Options that control the action of MQCLOSE */
MQLONG  CompCode; /* Completion code */
MQLONG  Reason;   /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCLOSE' USING HCONN, HOBJ, OPTIONS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN    PIC S9(9) BINARY.
** Object handle
01 HOBJ     PIC S9(9) BINARY.
** Options that control the action of MQCLOSE
01 OPTIONS  PIC S9(9) BINARY.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON   PIC S9(9) BINARY.
```

PL/I invocation

```
call MQCLOSE (Hconn, Hobj, Options, CompCode, Reason);
```

Declare the parameters as follows:

```
dc1 Hconn    fixed bin(31); /* Connection handle */
dc1 Hobj     fixed bin(31); /* Object handle */
dc1 Options  fixed bin(31); /* Options that control the action of
                             MQCLOSE */
dc1 CompCode fixed bin(31); /* Completion code */
dc1 Reason   fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
CALL MQCLOSE,(HCONN,HOBJ,OPTIONS,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN    DS F Connection handle
HOBJ     DS F Object handle
OPTIONS  DS F Options that control the action of MQCLOSE
COMPCODE DS F Completion code
REASON   DS F Reason code qualifying COMPCODE
```

TAL invocation

```
INT(32) .EXT HConn ;
INT(32) .EXT HObj;
INT(32) Options;
INT(32) .EXT CC;
INT(32) .EXT Reason;
```

```
CALL MQCLOSE(HConn, HObj, Options, CC, Reason);
```

Visual Basic invocation

```
MQCLOSE Hconn, Hobj, Options, CompCode, Reason
```

Declare the parameters as follows:

MQCLOSE – Language invocations

```
Dim Hconn    As Long 'Connection handle'  
Dim Hobj     As Long 'Object handle'  
Dim Options  As Long 'Options that control the action of MQCLOSE'  
Dim CompCode As Long 'Completion code'  
Dim Reason   As Long 'Reason code qualifying CompCode'
```

Chapter 30. MQCMIT – Commit changes

The MQCMIT call indicates to the queue manager that the application has reached a syncpoint, and that all of the message gets and puts that have occurred since the last syncpoint are to be made permanent. Messages put as part of a unit of work are made available to other applications; messages retrieved as part of a unit of work are deleted.

- On z/OS, the call is used only by batch programs (including IMS batch DL/I programs). On WebSphere Application Server embedded messaging using reduced function WebSphere MQ, this call can be used only by batch programs (**not** including IMS batch DL/I programs) when the unit of work affects only MQ resources.
- On OS/400, this call is not supported for applications running in compatibility mode.
- On Compaq NonStop Kernel, this call can be issued by the application but always returns completion code MQCC_FAILED and reason code MQRC_ENVIRONMENT_ERROR.
MQCMIT is valid for client applications to Compaq NonStop Kernel, and **must** be used to commit messages that have been put or got using MQPMO_SYNCPOINT or MQGMO_SYNCPOINT.
- On VSE/ESA, this call is used only by client programs and batch programs.

Syntax

MQCMIT (*Hconn*, *CompCode*, *Reason*)

Parameters

The MQCMIT call has the following parameters.

Hconn (MQHCONN) – input

Connection handle.

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_WARNING

Warning (partial completion).

MQCC_FAILED

Call failed.

MQCMIT – Reason parameter

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_BACKED_OUT

(2003, X'7D3') Unit of work backed out.

MQRC_OUTCOME_PENDING

(2124, X'84C') Result of commit operation is pending.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_SERV_LOAD_ERROR

(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR

(2374, X'946') API exit failed.

MQRC_ASID_MISMATCH

(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS

(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_STRUC_IN_USE

(2346, X'92A') Coupling-facility structure in use.

MQRC_CONNECTION_BROKEN

(2009, X'7D9') Connection to queue manager lost.

MQRC_ENVIRONMENT_ERROR

(2012, X'7DC') Call not valid in environment.

MQRC_HCONN_ERROR

(2018, X'7E2') Connection handle not valid.

MQRC_OBJECT_DAMAGED

(2101, X'835') Object damaged.

MQRC_OUTCOME_MIXED

(2123, X'84B') Result of commit or back-out operation is mixed.

MQRC_Q_MGR_STOPPING

(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM

(2102, X'836') Insufficient system resources available.

MQRC_STORAGE_MEDIUM_FULL

(2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE

(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR

(2195, X'893') Unexpected error occurred.

See Appendix A, "Return codes", on page 527 for more details.

Usage notes

1. This call can be used only when the queue manager itself coordinates the unit of work. This can be:
 - A local unit of work, where the changes affect only MQ resources.
 - A global unit of work, where the changes can affect resources belonging to other resource managers, as well as affecting MQ resources.

For further details about local and global units of work, see Chapter 28, “MQBEGIN – Begin unit of work”, on page 341.

2. In environments where the queue manager does not coordinate the unit of work, the appropriate commit call must be used instead of MQCMIT. The environment may also support an implicit commit caused by the application terminating normally.
 - On z/OS, the following calls should be used:
 - Batch programs (including IMS batch DL/I programs) can use the MQCMIT call if the unit of work affects only MQ resources. However, if the unit of work affects both MQ resources and resources belonging to other resource managers (for example, DB2), the SRRRCMIT call provided by the z/OS Recoverable Resource Service (RRS) should be used. The SRRRCMIT call commits changes to resources belonging to the resource managers that have been enabled for RRS coordination.

(On WebSphere Application Server embedded messaging using reduced function WebSphere MQ, this call can be used only by batch programs (**not** including IMS batch DL/I programs) when the unit of work affects only MQ resources.)
 - CICS applications should use the EXEC CICS SYNCPOINT command to commit the unit of work. Alternatively, ending the transaction results in an implicit commit of the unit of work. The MQCMIT call cannot be used for CICS applications.
 - IMS applications (other than batch DL/I programs) should use IMS calls such as GU and CHKP to commit the unit of work. The MQCMIT call cannot be used for IMS applications (other than batch DL/I programs).
 - On OS/400, this call can be used for local units of work coordinated by the queue manager. This means that a commitment definition must not exist at job level, that is, the STRCMTCTL command with the CMTSCOPE(*JOB) parameter must not have been issued for the job.
 - On Compaq NonStop Kernel, this call always returns a *CompCode* of MQCC_FAILED and a *Reason* of MQRC_ENVIRONMENT_ERROR. Transactions are managed externally through TM/MP.
 - On VSE/ESA, this call is used only by client programs and batch programs. In both cases the call causes the queue manager to issue the EXEC CICS SYNCPOINT command on behalf of the application.

This call is not supported for CICS applications, which should use instead the EXEC CICS SYNCPOINT command to cause changes to be committed. Changes are also committed if the application terminates normally; in this situation CICS executes an implicit SYNCPOINT on behalf of the application.
3. If an application ends with uncommitted changes in a unit of work, the disposition of those changes depends on whether the application ends normally or abnormally. See the usage notes in Chapter 33, “MQDISC – Disconnect queue manager”, on page 373 for further details.
4. When an application puts or gets messages in groups or segments of logical messages, the queue manager retains information relating to the message group and logical message for the last successful MQPUT and MQGET calls. This information is associated with the queue handle, and includes such things as:
 - The values of the *GroupId*, *MsgSeqNumber*, *Offset*, and *MsgFlags* fields in MQMD.
 - Whether the message is part of a unit of work.
 - For the MQPUT call: whether the message is persistent or nonpersistent.

MQCMIT – Usage notes

When a unit of work is committed, the queue manager retains the group and segment information, and the application can continue putting or getting messages in the current message group or logical message.

Retaining the group and segment information when a unit of work is committed allows the application to spread a large message group or large logical message consisting of many segments across several units of work. Using several units of work may be advantageous if the local queue manager has only limited queue storage. However, the application must maintain sufficient information to be able to restart putting or getting messages at the correct point in the event that a system failure occurs. For details of how to restart at the correct point after a system failure, see the MQPMO_LOGICAL_ORDER option described in Chapter 14, “MQPMO – Put-message options”, on page 229, and the MQGMO_LOGICAL_ORDER option described in Chapter 8, “MQGMO – Get-message options”, on page 95.

The remaining usage notes apply only when the queue manager coordinates the units of work:

5. A unit of work has the same scope as a connection handle. This means that all MQ calls which affect a particular unit of work must be performed using the same connection handle. Calls issued using a different connection handle (for example, calls issued by another application) affect a different unit of work. See the *Hconn* parameter described in MQCONN for information about the scope of connection handles.
6. Only messages that were put or retrieved as part of the current unit of work are affected by this call.
7. A long-running application that issues MQGET, MQPUT, or MQPUT1 calls within a unit of work, but which never issues a commit or back-out call, can cause queues to fill up with messages that are not available to other applications. To guard against this possibility, the administrator should set the *MaxUncommittedMsgs* queue-manager attribute to a value that is low enough to prevent runaway applications filling the queues, but high enough to allow the expected messaging applications to work correctly.
8. Note that in some environments, if the *Reason* parameter is MQRC_CONNECTION_BROKEN (with a *CompCode* of MQCC_FAILED), it is possible that the unit of work was successfully committed.

This applies to MQ client applications running in the following environments: Compaq OpenVMS Alpha, OS/2, Compaq NonStop Kernel, UNIX systems, and Windows.

Language invocations

The MQCMIT call is supported in the programming languages shown below.

C invocation

```
MQCMIT (Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;    /* Connection handle */
MQLONG  CompCode; /* Completion code */
MQLONG  Reason;   /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCMIT' USING HCONN, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN    PIC S9(9) BINARY.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON   PIC S9(9) BINARY.
```

PL/I invocation

```
call MQCMIT (Hconn, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn    fixed bin(31); /* Connection handle */
dcl CompCode fixed bin(31); /* Completion code */
dcl Reason   fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
CALL MQCMIT,(HCONN,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN      DS F Connection handle
COMPCODE   DS F Completion code
REASON     DS F Reason code qualifying COMPCODE
```

TAL invocation

```
INT(32)    .EXT Hconn;
INT(32)    .EXT CC;
INT(32)    .EXT Reason;
```

Visual Basic invocation

```
MQCMIT Hconn, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn    As Long 'Connection handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

MQCMIT – Language invocations

Chapter 31. MQCONN – Connect queue manager

The MQCONN call connects an application program to a queue manager. It provides a queue manager connection handle, which is used by the application on subsequent message queuing calls.

- On z/OS, CICS applications do not have to issue this call. These applications are connected automatically to the queue manager to which the CICS system is connected. However, the MQCONN and MQDISC calls are still accepted from CICS applications.
- On OS/400, applications running in compatibility mode do not have to issue this call. These applications are connected automatically to the queue manager when they issue the first MQOPEN call. However, the MQCONN and MQDISC calls are still accepted from OS/400 applications.

Other applications (that is, applications not running in compatibility mode) must use the MQCONN or MQCONNX call to connect to the queue manager, and the MQDISC call to disconnect from the queue manager. This is the recommended style of programming.

Syntax

MQCONN (*QMgrName*, *Hconn*, *CompCode*, *Reason*)

Parameters

The MQCONN call has the following parameters.

QMgrName (MQCHAR48) – input

Name of queue manager.

This is the name of the queue manager to which the application wishes to connect. The name can contain the following characters:

- Uppercase alphabetic characters (A through Z)
- Lowercase alphabetic characters (a through z)
- Numeric digits (0 through 9)
- Period (.), forward slash (/), underscore (_), percent (%)

The name must not contain leading or embedded blanks, but may contain trailing blanks. A null character can be used to indicate the end of significant data in the name; the null and any characters following it are treated as blanks. The following restrictions apply in the environments indicated:

- On systems that use EBCDIC Katakana, lowercase characters cannot be used.
- On z/OS, names that begin or end with an underscore cannot be processed by the operations and control panels. For this reason such names should be avoided.
- On OS/400, names containing lowercase characters, forward slash, or percent must be enclosed in quotation marks when specified on commands. These quotation marks must not be specified in the *QMgrName* parameter.

MQCONN – QMgrName parameter

If the name consists entirely of blanks, the name of the *default* queue manager is used.

The name specified for *QMgrName* must be the name of a *connectable* queue manager. The queue managers to which it is possible to connect are determined by the environment:

- On z/OS:
 - For CICS, you can use only the queue manager to which the CICS system is connected. The *QMgrName* parameter must still be specified, but its value is ignored; blanks are recommended.
 - For IMS, only queue managers which are listed in the subsystem definition table (CSQQDEFV), and listed in the SSM table in IMS, are connectable (see Usage note 6 on page 365).
 - For z/OS batch and TSO, only queue managers that reside on the same system as the application are connectable (see Usage note 6 on page 365).
- On VSE/ESA, you can use only the queue manager to which the CICS system is connected. The *QMgrName* parameter must still be specified; blanks are recommended.

Queue-sharing groups:

Note: This information does **not** apply when using the reduced function form of WebSphere MQ supplied with WebSphere Application Server.

On systems where several queue managers exist and are configured to form a queue-sharing group, the name of the queue-sharing group can be specified for *QMgrName* in place of the name of a queue manager. This allows the application to connect to *any* queue manager that is available in the queue-sharing group and that is on the same z/OS image as the application. The system can also be configured so that a blank *QMgrName* causes connection to the queue-sharing group instead of to the default queue manager.

If *QMgrName* specifies the name of the queue-sharing group, but there is also a queue manager with that name on the system, connection is made to the latter in preference to the former. Only if that connection fails is connection to one of the queue managers in the queue-sharing group attempted.

If the connection is successful, the handle returned by the MQCONN or MQCONNX call can be used to access *all* of the resources (both shared and nonshared) that belong to the particular queue manager to which connection has been made. Access to these resources is subject to the usual authorization controls.

If the application issues two MQCONN or MQCONNX calls in order to establish concurrent connections, and one or both calls specifies the name of the queue-sharing group, the second call may return completion code MQCC_WARNING and reason code MQRC_ALREADY_CONNECTED. This occurs when the second call connects to the same queue manager as the first call.

Queue-sharing groups are supported only on z/OS. Connection to a queue-sharing group is supported only in the batch, RRS batch, and TSO environments.

MQ client applications: For MQ client applications, a connection is attempted for each client-connection channel definition with the specified queue-manager name, until one is successful. The queue manager, however, must have the same name as the specified name. If an all-blank name is specified, each client-connection channel

MQCONN – QMgrName parameter

with an all-blank queue-manager name is tried until one is successful; in this case there is no check against the actual name of the queue manager.

MQ client applications are not supported in the following environments: z/OS, Windows 3.1, Windows 95, Windows 98. However, z/OS can act as an MQ server, to which MQ client applications can connect.

MQ client queue-manager groups: If the specified name starts with an asterisk (*), the actual queue manager to which connection is made may have a name that is different from that specified by the application. The specified name (without the asterisk) defines a *group* of queue managers that are eligible for connection. The implementation selects one from the group by trying each one in turn (in no defined order) until one is found to which a connection can be made. If none of the queue managers in the group is available for connection, the call fails. Each queue manager is tried once only. If an asterisk alone is specified for the name, an implementation-defined default queue-manager group is used.

Queue-manager groups are supported only for applications running in an MQ-client environment; the call fails if a non-client application specifies a queue-manager name beginning with an asterisk. A group is defined by providing several client connection channel definitions with the same queue-manager name (the specified name without the asterisk), to communicate with each of the queue managers in the group. The default group is defined by providing one or more client connection channel definitions, each with a blank queue-manager name (specifying an all-blank name therefore has the same effect as specifying a single asterisk for the name for a client application).

After connecting to one queue manager of a group, an application can specify blanks in the usual way in the queue-manager name fields in the message and object descriptors to mean the name of the queue manager to which the application has actually connected (the *local queue manager*). If the application needs to know this name, the MQINQ call can be issued to inquire the *QMGrName* queue-manager attribute.

Prefixing an asterisk to the connection name implies that the application is not dependent on connecting to a particular queue manager in the group. Suitable applications would be:

- Applications that put messages but do not get messages.
- Applications that put request messages and then get the reply messages from a *temporary dynamic* queue.

Unsuitable applications would be those that need to get messages from a particular queue at a particular queue manager; such applications should not prefix the name with an asterisk.

Note that if an asterisk is specified, the maximum length of the remainder of the name is 47 characters.

Queue-manager groups are not supported in the following environments: z/OS, Windows 3.1, Windows 95, Windows 98.

The length of this parameter is given by MQ_Q_MGR_NAME_LENGTH.

Hconn (MQHCONN) – output

Connection handle.

MQCONN – Hconn parameter

This handle represents the connection to the queue manager. It must be specified on all subsequent message queuing calls issued by the application. It ceases to be valid when the MQDISC call is issued, or when the unit of processing that defines the scope of the handle terminates.

Handle scope: The scope of the handle returned depends on the call used to connect to the queue manager (MQCONN or MQCONNX). If the call used is MQCONNX, the scope of the handle also depends on the MQCNO_HANDLE_SHARE_* option specified in the *Options* field of the MQCNO structure.

- If the call is MQCONN, or the MQCNO_HANDLE_SHARE_NONE option is specified, the handle returned is a *nonshared* handle.

The scope of a nonshared handle is the smallest unit of parallel processing supported by the platform on which the application is running (see below for details); the handle is not valid outside the unit of parallel processing from which the call was issued.

- If the MQCNO_HANDLE_SHARE_BLOCK or MQCNO_HANDLE_SHARE_NO_BLOCK option is specified, the handle returned is a *shared* handle.

The scope of a shared handle is the process that owns the thread from which the call was issued; the handle can be used from any thread that belongs to that process. Not all platforms support threads.

The scope of nonshared handles on various platforms is shown in Table 85.

Table 85. Scope of nonshared handles on various platforms

Platform	Scope of nonshared handle
Compaq OpenVMS Alpha	Thread
PC DOS	System
z/OS	<ul style="list-style-type: none">• CICS: the CICS task• IMS: the task, up to the next syncpoint (excludes subtasks of the task)• z/OS batch and TSO: the task (excludes subtasks of the task)
OS/2	Thread
OS/400	Job
Compaq NonStop Kernel	Process
UNIX systems	Thread
VSE/ESA	CICS transaction
16-bit Windows applications	Process
32-bit Windows applications	Thread

On z/OS for CICS applications, and on OS/400 for applications running in compatibility mode, the value returned is:

MQHC_DEF_HCONN

Default connection handle.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_ALREADY_CONNECTED
(2002, X'7D2') Application already connected.

MQRC_CLUSTER_EXIT_LOAD_ERROR
(2267, X'8DB') Unable to load cluster workload exit.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_CONN_LOAD_ERROR
(2129, X'851') Unable to load adapter connection module.

MQRC_ADAPTER_DEFS_ERROR
(2131, X'853') Adapter subsystem definition module not valid.

MQRC_ADAPTER_DEFS_LOAD_ERROR
(2132, X'854') Unable to load adapter subsystem definition module.

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ADAPTER_STORAGE_SHORTAGE
(2127, X'84F') Insufficient storage for adapter.

MQRC_ANOTHER_Q_MGR_CONNECTED
(2103, X'837') Another queue manager already connected.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_INIT_ERROR
(2375, X'947') API exit initialization failed.

MQRC_API_EXIT_TERM_ERROR
(2376, X'948') API exit termination failed.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CONN_ID_IN_USE
(2160, X'870') Connection identifier already in use.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_ERROR
(2273, X'8E1') Error processing MQCONN call.

MQRC_CONNECTION_QUIESCING
(2202, X'89A') Connection quiescing.

MQCONN – Reason parameter

	MQRC_CONNECTION_STOPPING (2203, X'89B')	Connection shutting down.
	MQRC_CRYPTO_HARDWARE_ERROR (2382, X'94E')	Cryptographic hardware configuration error.
	MQRC_DUPLICATE_RECOV_COORD (2163, X'873')	Recovery coordinator already exists.
	MQRC_ENVIRONMENT_ERROR (2012, X'7DC')	Call not valid in environment.
	MQRC_HCONN_ERROR (2018, X'7E2')	Connection handle not valid.
	MQRC_KEY_REPOSITORY_ERROR (2381, X'94D')	Key repository not valid.
	MQRC_MAX_CONNS_LIMIT_REACHED (2025, X'7E9')	Maximum number of connections reached.
	MQRC_NOT_AUTHORIZED (2035, X'7F3')	Not authorized for access.
	MQRC_OPEN_FAILED (2137, X'859')	Object not opened successfully.
	MQRC_Q_MGR_NAME_ERROR (2058, X'80A')	Queue manager name not valid or not known.
	MQRC_Q_MGR_NOT_AVAILABLE (2059, X'80B')	Queue manager not available for connection.
	MQRC_Q_MGR QUIESCING (2161, X'871')	Queue manager quiescing.
	MQRC_Q_MGR_STOPPING (2162, X'872')	Queue manager shutting down.
	MQRC_RESOURCE_PROBLEM (2102, X'836')	Insufficient system resources available.
	MQRC_SECURITY_ERROR (2063, X'80F')	Security error occurred.
	MQRC_SSL_ALREADY_INITIALIZED (2391, X'957')	SSL already initialized.
	MQRC_SSL_INITIALIZATION_ERROR (2393, X'959')	SSL initialization error.
	MQRC_STORAGE_NOT_AVAILABLE (2071, X'817')	Insufficient storage available.
	MQRC_UNEXPECTED_ERROR (2195, X'893')	Unexpected error occurred.

For more information on these reason codes, see Appendix A, “Return codes”, on page 527.

Usage notes

1. The queue manager to which connection is made using the MQCONN call is called the *local queue manager*.
2. Queues that are owned by the local queue manager appear to the application as local queues. It is possible to put messages on and get messages from these queues.
Shared queues that are owned by the queue-sharing group to which the local queue manager belongs appear to the application as local queues. It is possible to put messages on and get messages from these queues.
Queues that are owned by remote queue managers appear as remote queues. It is possible to put messages on these queues, but not possible to get messages from these queues.

3. If the queue manager fails while an application is running, the application must issue the MQCONN call again in order to obtain a new connection handle to use on subsequent MQ calls. The application can issue the MQCONN call periodically until the call succeeds.

If an application is not sure whether it is connected to the queue manager, the application can safely issue an MQCONN call in order to obtain a connection handle. If the application is already connected, the handle returned is the same as that returned by the previous MQCONN call, but with completion code MQCC_WARNING and reason code MQRC_ALREADY_CONNECTED.

4. When the application has finished using MQ calls, the application should use the MQDISC call to disconnect from the queue manager.
5. On z/OS:

- Batch, TSO, and IMS applications must issue the MQCONN call in order to be able to use the other MQ calls. These applications can connect to more than one queue manager concurrently.

If the queue manager fails, the application must issue the call again after the queue manager has restarted in order to obtain a new connection handle.

Although IMS applications can issue the MQCONN call repeatedly, even when already connected, this is not recommended for online message processing programs (MPPs).

- CICS applications do not have to issue the MQCONN call in order to be able to use the other MQ calls, but can do so if they wish; both the MQCONN call and the MQDISC call are accepted. However, it is not possible to connect to more than one queue manager concurrently.

If the queue manager fails, these applications are automatically reconnected when the queue manager restarts, and so do not need to issue the MQCONN call.

6. On z/OS, to define the available queue managers:

- For batch applications, system programmers can use the CSQBDEF macro to create a module (CSQBDEFV) that defines the default queue-manager name, or queue-sharing group name.
- For IMS applications, system programmers can use the CSQQDEFX macro to create a module (CSQQDEFV) that defines the names of the available queue managers and specifies the default queue manager.

In addition, each a queue manager must be defined to the IMS control region and to each dependent region accessing that queue manager. To do this, you must create a subsystem member in the IMS.PROCLIB library and identify the subsystem member to the applicable IMS regions. If an application attempts to connect to a queue manager that is not defined in the subsystem member for its IMS region, the application abends.

For more information on using these macros, see the *WebSphere MQ for z/OS System Setup Guide*.

7. On OS/400, applications written for previous releases of the queue manager can run without the need for recompilation. This is called *compatibility mode*. This mode of operation provides a compatible run-time environment for applications. It comprises the following:

- The service program AMQZSTUB residing in the library QMQM.

AMQZSTUB provides the same public interface as previous releases, and has the same signature. This service program can be used to access the MQI through bound procedure calls.

- The program QMQM residing in the library QMQM.

MQCONN – Usage notes

QMQM provides a means of accessing the MQI through dynamic program calls.

- Programs MQCLOSE, MQCONN, MQDISC, MQGET, MQINQ, MQOPEN, MQPUT, MQPUT1, and MQSET residing in the library QMQM.

These programs also provide a means of accessing the MQI through dynamic program calls, but with a parameter list that corresponds to the standard descriptions of the MQ calls.

These three interfaces do not include capabilities that were introduced in version 5.1. For example, the MQBACK, MQCMIT, and MQCONNX calls are not supported. The support provided by these interfaces is for single-threaded applications only.

Support for the new MQ calls in single-threaded applications, and for all MQ calls in multi-threaded applications, is provided through the service programs LIBMQM and LIBMQM_R respectively.

8. On OS/400, programs that end abnormally are not automatically disconnected from the queue manager. Therefore applications should be written to allow for the possibility of the MQCONN or MQCONNX call returning completion code MQCC_WARNING and reason code MQRC_ALREADY_CONNECTED. The connection handle returned in this situation can be used as normal.

Language invocations

The MQCONN call is supported in the programming languages shown below.

C invocation

```
MQCONN (QMgrName, &Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR48 QMgrName; /* Name of queue manager */
MQHCONN  Hconn;    /* Connection handle */
MQLONG   CompCode; /* Completion code */
MQLONG   Reason;   /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCONN' USING QMGRNAME, HCONN, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Name of queue manager
01 QMGRNAME PIC X(48).
** Connection handle
01 HCONN PIC S9(9) BINARY.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON PIC S9(9) BINARY.
```

PL/I invocation

```
call MQCONN (QMgrName, Hconn, CompCode, Reason);
```

Declare the parameters as follows:

MQCONN – Language invocations

```
dc1 QMgrName char(48);      /* Name of queue manager */
dc1 Hconn    fixed bin(31); /* Connection handle */
dc1 CompCode fixed bin(31); /* Completion code */
dc1 Reason   fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
CALL MQCONN,(QMGRNAME,HCONN,COMPCODE,REASON)
```

Declare the parameters as follows:

```
QMGRNAME DS CL48 Name of queue manager
HCONN     DS F    Connection handle
COMPCODE  DS F    Completion code
REASON    DS F    Reason code qualifying COMPCODE
```

TAL invocation

```
STRING .EXT InQMgr[0:47];
INT(32) .EXT HConn ;
INT(32) .EXT CC;
INT(32) .EXT Reason;

CALL MQCONN(InQMgr, HConn, CC, Reason);
```

Visual Basic invocation

```
MQCONN QMgrName, Hconn, CompCode, Reason
```

Declare the parameters as follows:

```
Dim QMgrName As String*48 'Name of queue manager'
Dim Hconn    As Long      'Connection handle'
Dim CompCode As Long      'Completion code'
Dim Reason   As Long      'Reason code qualifying CompCode'
```

MQCONN – Language invocations

Chapter 32. MQCONNX – Connect queue manager (extended)

The MQCONNX call connects an application program to a queue manager. It provides a queue manager connection handle, which is used by the application on subsequent MQ calls.

The MQCONNX call is similar to the MQCONN call, except that MQCONNX allows options to be specified to control the way that the call works.

- This call is supported in the following environments: AIX, HP-UX, z/OS, OS/2, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.
- On OS/400, this call is not supported for applications running in compatibility mode.

Syntax

MQCONNX (*QMgrName*, *ConnectOpts*, *Hconn*, *CompCode*, *Reason*)

Parameters

The MQCONNX call has the following parameters.

QMgrName (MQCHAR48) – input

Name of queue manager.

See the *QMgrName* parameter described in Chapter 31, “MQCONN – Connect queue manager”, on page 359 for details.

ConnectOpts (MQCNO) – input/output

Options that control the action of MQCONNX.

See Chapter 5, “MQCNO – Connect options”, on page 61 for details.

Hconn (MQHCONN) – output

Connection handle.

See the *Hconn* parameter described in Chapter 31, “MQCONN – Connect queue manager”, on page 359 for details.

CompCode (MQLONG) – output

Completion code.

See the *CompCode* parameter described in Chapter 31, “MQCONN – Connect queue manager”, on page 359 for details.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

MQCONN – Reason parameter

See the *Reason* parameter described in Chapter 31, “MQCONN – Connect queue manager”, on page 359 for details of possible reason codes.

The following additional reason codes can be returned by the MQCONN call:

If *CompCode* is MQCC_FAILED:

| **MQRC_AIR_ERROR**

| (2385, X'951') Authentication information record not valid.

| **MQRC_API_EXIT_ERROR**

| (2374, X'946') An API exit function returned an invalid response code, or
| failed in some other way.

| **MQRC_API_EXIT_INIT_ERROR**

| (2375, X'947') The queue manager encountered an error while attempting
| to initialize the execution environment for an API exit function.

| **MQRC_API_EXIT_TERM_ERROR**

| (2376, X'948') The queue manager encountered an error while attempting
| to terminate the execution environment for an API exit function.

| **MQRC_AUTH_INFO_CONN_NAME_ERROR**

| (2387, X'953') Authentication information connection name not valid.

| **MQRC_AUTH_INFO_REC_COUNT_ERROR**

| (2383, X'94F') Authentication information record count not valid.

| **MQRC_AUTH_INFO_REC_ERROR**

| (2384, X'950') Authentication information record fields not valid.

| **MQRC_AUTH_INFO_TYPE_ERROR**

| (2386, X'952') Authentication information type not valid.

| **MQRC_CD_ERROR**

| (2277, X'8E5') Channel definition not valid.

| **MQRC_CLIENT_CONN_ERROR**

| (2278, X'8E6') Client connection fields not valid.

| **MQRC_CNO_ERROR**

| (2139, X'85B') Connect-options structure not valid.

| **MQRC_CONN_TAG_IN_USE**

| (2271, X'8DF') Connection tag in use.

| **MQRC_CONN_TAG_NOT_USABLE**

| (2350, X'92E') Connection tag not usable.

| **MQRC_CRYPTO_HARDWARE_ERROR**

| (2382, X'94E') Cryptographic hardware configuration error.

| **MQRC_KEY_REPOSITORY_ERROR**

| (2381, X'94D') Key repository not valid.

| **MQRC_LDAP_PASSWORD_ERROR**

| (2390, X'956') LDAP password not valid.

| **MQRC_LDAP_USER_NAME_ERROR**

| (2388, X'954') LDAP user name fields not valid.

| **MQRC_LDAP_USER_NAME_LENGTH_ERR**

| (2389, X'955') LDAP user name length not valid.

| **MQRC_OPTIONS_ERROR**

| (2046, X'7FE') Options not valid or not consistent.

| **MQRC_SCO_ERROR**

| (2380, X'94C') SSL configuration options structure not valid.

| **MQRC_SSL_ALREADY_INITIALIZED**

| (2391, X'957') SSL already initialized.

| **MQRC_SSL_CONFIG_ERROR**

| (2392, X'958') SSL configuration error.

| **MQRC_SSL_INITIALIZATION_ERROR**

| (2393, X'959') SSL initialization error.

For more information on these reason codes, see Appendix A, “Return codes”, on page 527.

Usage notes

1. For the Visual Basic programming language, the following point applies:
 - The *ConnectOpts* parameter is declared as being of type MQCNO. If the application is running as an WebSphere MQ client, and wishes to specify the parameters of the client-connection channel, the MQCONNXAny call should be used in place of MQCONNX, so that the application can specify an MQCNOCD structure on the call in place of an MQCNO structure.

The MQCONNXAny call has the same parameters as the MQCONNX call, except that the *ConnectOpts* parameter is declared as being of type Any, allowing either an MQCNO structure or an MQCNOCD structure to be specified as the parameter. However, this means that the *ConnectOpts* parameter cannot be checked to ensure that it is the correct data type.

Language invocations

The MQCONNX call is supported in the programming languages shown below.

C invocation

```
MQCONNX (QMgrName, &ConnectOpts, &Hconn, &CompCode,
         &Reason);
```

Declare the parameters as follows:

```
MQCHAR48 QMgrName;    /* Name of queue manager */
MQCNO    ConnectOpts; /* Options that control the action of MQCONNX */
MQHCONN  Hconn;      /* Connection handle */
MQLONG   CompCode;   /* Completion code */
MQLONG   Reason;     /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCONNX' USING QMGRNAME, CONNECTOPTS, HCONN, COMPCODE,
                    REASON.
```

Declare the parameters as follows:

```
** Name of queue manager
01 QMGRNAME    PIC X(48).
** Options that control the action of MQCONNX
01 CONNECTOPTS.
   COPY CMQCNOV.
** Connection handle
01 HCONN      PIC S9(9) BINARY.
** Completion code
01 COMPCODE   PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON     PIC S9(9) BINARY.
```

PL/I invocation

```
call MQCONNX (QMgrName, ConnectOpts, Hconn, CompCode, Reason);
```

Declare the parameters as follows:

```
dc1 QMgrName   char(48);    /* Name of queue manager */
dc1 ConnectOpts like MQCNO; /* Options that control the action of
                             MQCONNX */
```

MQCONN – Language invocations

```
dc1 Hconn          fixed bin(31); /* Connection handle */
dc1 CompCode       fixed bin(31); /* Completion code */
dc1 Reason         fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
CALL MQCONN,(QMGRNAME,CONNECTOPTS,HCONN,COMPCODE,REASON)
```

Declare the parameters as follows:

QMGRNAME	DS	CL48	Name of queue manager
CONNECTOPTS	CMQCNOA	,	Options that control the action of MQCONN
HCONN	DS	F	Connection handle
COMPCODE	DS	F	Completion code
REASON	DS	F	Reason code qualifying COMPCODE

Visual Basic invocation

```
MQCONN QMgrName, ConnectOpts, Hconn, CompCode, Reason
```

Declare the parameters as follows:

```
Dim QMgrName As String*48 'Name of queue manager'
Dim ConnectOpts As MQCNO 'Options that control the action of'
                          'MQCONN'
Dim Hconn As Long 'Connection handle'
Dim CompCode As Long 'Completion code'
Dim Reason As Long 'Reason code qualifying CompCode'
```

Chapter 33. MQDISC – Disconnect queue manager

The MQDISC call breaks the connection between the queue manager and the application program, and is the inverse of the MQCONN or MQCONNX call.

- On z/OS, CICS applications do not need to issue this call to disconnect from the queue manager, but may need to issue it in order to end the use of a connection tag.
- On OS/400, applications running in compatibility mode do not need to issue this call. See Chapter 31, “MQCONN – Connect queue manager”, on page 359 for more information.

Syntax

MQDISC (*Hconn*, *CompCode*, *Reason*)

Parameters

The MQDISC call has the following parameters.

Hconn (MQHCONN) – input/output

Connection handle.

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on OS/400 for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN

Default connection handle.

On successful completion of the call, the queue manager sets *Hconn* to a value that is not a valid handle for the environment. This value is:

MQHC_UNUSABLE_HCONN

Unusable connection handle.

On z/OS, *Hconn* is set to a value which is undefined.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_WARNING

Warning (partial completion).

MQCC_FAILED

Call failed.

MQDISC – Reason parameter

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_BACKED_OUT

(2003, X'7D3') Unit of work backed out.

MQRC_CONN_TAG_NOT_RELEASED

(2344, X'928') Connection tag not released.

MQRC_OUTCOME_PENDING

(2124, X'84C') Result of commit operation is pending.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_DISC_LOAD_ERROR

(2138, X'85A') Unable to load adapter disconnection module.

MQRC_ADAPTER_NOT_AVAILABLE

(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR

(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR

(2374, X'946') API exit failed.

MQRC_API_EXIT_INIT_ERROR

(2375, X'947') API exit initialization failed.

MQRC_API_EXIT_TERM_ERROR

(2376, X'948') API exit termination failed.

MQRC_ASID_MISMATCH

(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS

(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CONNECTION_BROKEN

(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_STOPPING

(2203, X'89B') Connection shutting down.

MQRC_HCONN_ERROR

(2018, X'7E2') Connection handle not valid.

MQRC_OUTCOME_MIXED

(2123, X'84B') Result of commit or back-out operation is mixed.

MQRC_PAGESET_ERROR

(2193, X'891') Error accessing page-set data set.

MQRC_Q_MGR_NAME_ERROR

(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE

(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_STOPPING

(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM

(2102, X'836') Insufficient system resources available.

MQRC_STORAGE_NOT_AVAILABLE

(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR

(2195, X'893') Unexpected error occurred.

For more information on these reason codes, see Appendix A, “Return codes”, on page 527.

Usage notes

1. If an MQDISC call is issued when the connection still has objects open under that connection,, those objects are closed by the queue manager, with the close options set to MQCO_NONE.
2. If the application ends with uncommitted changes in a unit of work, the disposition of those changes depends on how the application ends:
 - a. If the application issues the MQDISC call before ending:
 - For a queue-manager-coordinated unit of work, the queue manager issues the MQCMIT call on behalf of the application. The unit of work is committed if possible, and backed out if not.
 - On z/OS, batch programs (including IMS batch DL/1 programs) are like this.
 - For an externally-coordinated unit of work, there is no change in the status of the unit of work; however, the queue manager will typically indicate that the unit of work should be committed, when asked by the unit-of-work coordinator.
 - On z/OS, CICS, IMS (other than batch DL/1 programs), and RRS applications are like this.
 - b. If the application ends normally but without issuing the MQDISC call, the action taken depends on the environment:
 - On z/OS, and on VSE/ESA for CICS applications, the actions described under (a) above occur.
 - In all other cases, the actions described under (c) below occur.

Because of the differences between environments, applications which are intended to be portable should ensure that the unit of work is committed or backed out before the application ends.
 - c. If the application ends *abnormally* without issuing the MQDISC call, the unit of work is backed out.
3. On z/OS, the following points apply:
 - CICS applications do not have to issue the MQDISC call to disconnect from the queue manager. This is because the CICS system itself connects to the queue manager, and the MQDISC call has no effect on this connection.
 - CICS, IMS (other than batch DL/1 programs), and RRS applications use units of work that are coordinated by an external unit-of-work coordinator. As a result, the MQDISC call does not affect the status of the unit of work (if any) that exists when the call is issued.

However the MQDISC call *does* indicate the end of use of the connection tag *ConnTag* that was associated with the connection by an earlier MQCONN call issued by the application. If there is a active unit of work that references the connection tag when the MQDISC call is issued, the call completes with completion code MQCC_WARNING and reason code MQRC_CONN_TAG_NOT_RELEASED. The connection tag does not become available for reuse until the external unit-of-work coordinator has resolved the unit of work.
4. On OS/2, if an application terminates a thread without first issuing MQDISC, and a new thread is subsequently created within the same process, and that thread issues message-queuing calls, the behavior of the queue manager is undefined.

MQDISC – Usage notes

5. On OS/400, applications running in compatibility mode do not have to issue this call; see the MQCONN call for more details.
6. On Compaq NonStop Kernel, an implicit syncpoint does not occur if a queue manager coordinated unit of work is in progress when MQDISC is called. This is because the NSK queue manager cannot be a coordinator of a unit of work. Coordination is provided by the TM/MP subsystem.

Language invocations

The MQDISC call is supported in the programming languages shown below.

C invocation

```
MQDISC (&Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;    /* Connection handle */
MQLONG   CompCode; /* Completion code */
MQLONG   Reason;   /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQDISC' USING HCONN, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN    PIC S9(9) BINARY.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON   PIC S9(9) BINARY.
```

PL/I invocation

```
call MQDISC (Hconn, CompCode, Reason);
```

Declare the parameters as follows:

```
dc1 Hconn    fixed bin(31); /* Connection handle */
dc1 CompCode fixed bin(31); /* Completion code */
dc1 Reason   fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
CALL MQDISC,(HCONN,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN      DS F Connection handle
COMPCODE   DS F Completion code
REASON     DS F Reason code qualifying COMPCODE
```

TAL invocation

```
INT(32)    .EXT HConn ;
INT(32)    .EXT CC;
INT(32)    .EXT Reason;
```

```
CALL MQDISC(HConn, CC, Reason);
```

Visual Basic invocation

MQDISC Hconn, CompCode, Reason

Declare the parameters as follows:

```
Dim Hconn    As Long 'Connection handle'  
Dim CompCode As Long 'Completion code'  
Dim Reason  As Long 'Reason code qualifying CompCode'
```

MQDISC – Language invocations

Chapter 34. MQGET – Get message

The MQGET call retrieves a message from a local queue that has been opened using the MQOPEN call.

Syntax

MQGET (*Hconn*, *Hobj*, *MsgDesc*, *GetMsgOpts*, *BufferLength*,
Buffer, *DataLength*, *CompCode*, *Reason*)

Parameters

The MQGET call has the following parameters.

Hconn (MQHCONN) – input

Connection handle.

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on OS/400 for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Hobj (MQHOBJ) – input

Object handle.

This handle represents the queue from which a message is to be retrieved. The value of *Hobj* was returned by a previous MQOPEN call. The queue must have been opened with one or more of the following options (see Chapter 36, “MQOPEN – Open object”, on page 405 for details):

MQOO_INPUT_SHARED
MQOO_INPUT_EXCLUSIVE
MQOO_INPUT_AS_Q_DEF
MQOO_BROWSE

MsgDesc (MQMD) – input/output

Message descriptor.

This structure describes the attributes of the message required, and the attributes of the message retrieved. See Chapter 10, “MQMD – Message descriptor”, on page 141 for details.

If *BufferLength* is less than the message length, *MsgDesc* is still filled in by the queue manager, whether or not MQGMO_ACCEPT_TRUNCATED_MSG is specified on the *GetMsgOpts* parameter (see the *Options* field described in Chapter 8, “MQGMO – Get-message options”, on page 95).

MQGET – MsgDesc parameter

If the application provides a version-1 MQMD, the message returned has an MQMDE prefixed to the application message data, but *only* if one or more of the fields in the MQMDE has a nondefault value. If all of the fields in the MQMDE have default values, the MQMDE is omitted. A format name of MQFMT_MD_EXTENSION in the *Format* field in MQMD indicates that an MQMDE is present.

GetMsgOpts (MQGMO) – input/output

Options that control the action of MQGET.

See Chapter 8, “MQGMO – Get-message options”, on page 95 for details.

BufferLength (MQLONG) – input

Length in bytes of the *Buffer* area.

Zero can be specified for messages that have no data, or if the message is to be removed from the queue and the data discarded (MQGMO_ACCEPT_TRUNCATED_MSG must be specified in this case).

Note: The length of the longest message that it is possible to read from the queue is given by the *MaxMsgLength* queue attribute; see Chapter 40, “Attributes for queues”, on page 457.

Buffer (MQBYTE×BufferLength) – output

Area to contain the message data.

The buffer should be aligned on a boundary appropriate to the nature of the data in the message. 4-byte alignment should be suitable for most messages (including messages containing MQ header structures), but some messages may require more stringent alignment. For example, a message containing a 64-bit binary integer might require 8-byte alignment.

If *BufferLength* is less than the message length, as much of the message as possible is moved into *Buffer*; this happens whether or not MQGMO_ACCEPT_TRUNCATED_MSG is specified on the *GetMsgOpts* parameter (see the *Options* field described in Chapter 8, “MQGMO – Get-message options”, on page 95 for more information).

The character set and encoding of the data in *Buffer* are given (respectively) by the *CodedCharSetId* and *Encoding* fields returned in the *MsgDesc* parameter. If these are different from the values required by the receiver, the receiver must convert the application message data to the character set and encoding required. The MQGMO_CONVERT option can be used (with a user-written exit if necessary) to perform the conversion of the message data; see Chapter 8, “MQGMO – Get-message options”, on page 95 for details of this option.

Note: All of the other parameters on the MQGET call are in the character set and encoding of the local queue manager (given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE, respectively).

If the call fails, the contents of the buffer may still have changed.

In the C programming language, the parameter is declared as a pointer-to-void; this means that the address of any type of data can be specified as the parameter.

If the *BufferLength* parameter is zero, *Buffer* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler can be null.

DataLength (MQLONG) – output

Length of the message.

This is the length in bytes of the application data *in the message*. If this is greater than *BufferLength*, only *BufferLength* bytes are returned in the *Buffer* parameter (that is, the message is truncated). If the value is zero, it means that the message contains no application data.

If *BufferLength* is less than the message length, *DataLength* is still filled in by the queue manager, whether or not MQGMO_ACCEPT_TRUNCATED_MSG is specified on the *GetMsgOpts* parameter (see the *Options* field described in Chapter 8, “MQGMO – Get-message options”, on page 95 for more information). This allows the application to determine the size of the buffer required to accommodate the message data, and then reissue the call with a buffer of the appropriate size.

However, if the MQGMO_CONVERT option is specified, and the converted message data is too long to fit in *Buffer*, the value returned for *DataLength* is:

- The length of the *unconverted* data, for queue-manager defined formats.
In this case, if the nature of the data causes it to expand during conversion, the application must allocate a buffer somewhat bigger than the value returned by the queue manager for *DataLength*.
- The value returned by the data-conversion exit, for application-defined formats.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

The reason codes listed below are the ones that the queue manager can return for the *Reason* parameter. If the application specifies the MQGMO_CONVERT option, and a user-written exit is invoked to convert some or all of the message data, it is the exit that decides what value is returned for the *Reason* parameter. As a result, values other than those documented below are possible.

If *CompCode* is MQCC_OK :

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQGET – Reason parameter

MQRC_CONVERTED_MSG_TOO_BIG	(2120, X'848')	Converted data too big for buffer.
MQRC_CONVERTED_STRING_TOO_BIG	(2190, X'88E')	Converted string too big for field.
MQRC_DBCS_ERROR	(2150, X'866')	DBCS string not valid.
MQRC_FORMAT_ERROR	(2110, X'83E')	Message format not valid.
MQRC_INCOMPLETE_GROUP	(2241, X'8C1')	Message group not complete.
MQRC_INCOMPLETE_MSG	(2242, X'8C2')	Logical message not complete.
MQRC_INCONSISTENT_CCSDS	(2243, X'8C3')	Message segments have differing CCSIDs.
MQRC_INCONSISTENT_ENCODINGS	(2244, X'8C4')	Message segments have differing encodings.
MQRC_INCONSISTENT_UOW	(2245, X'8C5')	Inconsistent unit-of-work specification.
MQRC_NO_MSG_LOCKED	(2209, X'8A1')	No message locked.
MQRC_NOT_CONVERTED	(2119, X'847')	Message data not converted.
MQRC_PARTIALLY_CONVERTED	(2272, X'8E0')	Message data partially converted.
MQRC_SIGNAL_REQUEST_ACCEPTED	(2070, X'816')	No message returned (but signal request accepted).
MQRC_SOURCE_BUFFER_ERROR	(2145, X'861')	Source buffer parameter not valid.
MQRC_SOURCE_CCSDS_ERROR	(2111, X'83F')	Source coded character set identifier not valid.
MQRC_SOURCE_DECIMAL_ENC_ERROR	(2113, X'841')	Packed-decimal encoding in message not recognized.
MQRC_SOURCE_FLOAT_ENC_ERROR	(2114, X'842')	Floating-point encoding in message not recognized.
MQRC_SOURCE_INTEGER_ENC_ERROR	(2112, X'840')	Source integer encoding not recognized.
MQRC_SOURCE_LENGTH_ERROR	(2143, X'85F')	Source length parameter not valid.
MQRC_TARGET_BUFFER_ERROR	(2146, X'862')	Target buffer parameter not valid.
MQRC_TARGET_CCSDS_ERROR	(2115, X'843')	Target coded character set identifier not valid.
MQRC_TARGET_DECIMAL_ENC_ERROR	(2117, X'845')	Packed-decimal encoding specified by receiver not recognized.
MQRC_TARGET_FLOAT_ENC_ERROR	(2118, X'846')	Floating-point encoding specified by receiver not recognized.
MQRC_TARGET_INTEGER_ENC_ERROR	(2116, X'844')	Target integer encoding not recognized.
MQRC_TRUNCATED_MSG_ACCEPTED	(2079, X'81F')	Truncated message returned (processing completed).
MQRC_TRUNCATED_MSG_FAILED	(2080, X'820')	Truncated message returned (processing not completed).

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_CONV_LOAD_ERROR
(2133, X'855') Unable to load data conversion services modules.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BACKED_OUT
(2003, X'7D3') Unit of work backed out.

MQRC_BUFFER_ERROR
(2004, X'7D4') Buffer parameter not valid.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_STRUC_FAILED
(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CF_STRUC_LIST_HDR_IN_USE
(2347, X'92B') Coupling-facility structure list-header in use.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_CORREL_ID_ERROR
(2207, X'89F') Correlation-identifier error.

MQRC_DATA_LENGTH_ERROR
(2010, X'7DA') Data length parameter not valid.

MQRC_GET_INHIBITED
(2016, X'7E0') Gets inhibited for the queue.

MQRC_GLOBAL_UOW_CONFLICT
(2351, X'92F') Global units of work conflict.

MQRC_GMO_ERROR
(2186, X'88A') Get-message options structure not valid.

MQRC_HANDLE_IN_USE_FOR_UOW
(2353, X'931') Handle in use for global unit of work.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_INCONSISTENT_BROWSE
(2259, X'8D3') Inconsistent browse specification.

MQRC_INCONSISTENT_UOW
(2245, X'8C5') Inconsistent unit-of-work specification.

MQGET – Reason parameter

MQRC_INVALID_MSG_UNDER_CURSOR	(2246, X'8C6')	Message under cursor not valid for retrieval.
MQRC_LOCAL_UOW_CONFLICT	(2352, X'930')	Global unit of work conflicts with local unit of work.
MQRC_MATCH_OPTIONS_ERROR	(2247, X'8C7')	Match options not valid.
MQRC_MD_ERROR	(2026, X'7EA')	Message descriptor not valid.
MQRC_MSG_ID_ERROR	(2206, X'89E')	Message-identifier error.
MQRC_MSG_SEQ_NUMBER_ERROR	(2250, X'8CA')	Message sequence number not valid.
MQRC_MSG_TOKEN_ERROR	(2331, X'91B')	Use of message token not valid.
MQRC_NO_MSG_AVAILABLE	(2033, X'7F1')	No message available.
MQRC_NO_MSG_UNDER_CURSOR	(2034, X'7F2')	Browse cursor not positioned on message.
MQRC_NOT_OPEN_FOR_BROWSE	(2036, X'7F4')	Queue not open for browse.
MQRC_NOT_OPEN_FOR_INPUT	(2037, X'7F5')	Queue not open for input.
MQRC_OBJECT_CHANGED	(2041, X'7F9')	Object definition changed since opened.
MQRC_OBJECT_DAMAGED	(2101, X'835')	Object damaged.
MQRC_OPTIONS_ERROR	(2046, X'7FE')	Options not valid or not consistent.
MQRC_PAGESET_ERROR	(2193, X'891')	Error accessing page-set data set.
MQRC_Q_DELETED	(2052, X'804')	Queue has been deleted.
MQRC_Q_INDEX_TYPE_ERROR	(2394, X'95A')	Queue has wrong index type.
MQRC_Q_MGR_NAME_ERROR	(2058, X'80A')	Queue manager name not valid or not known.
MQRC_Q_MGR_NOT_AVAILABLE	(2059, X'80B')	Queue manager not available for connection.
MQRC_Q_MGR QUIESCING	(2161, X'871')	Queue manager quiescing.
MQRC_Q_MGR STOPPING	(2162, X'872')	Queue manager shutting down.
MQRC_RESOURCE PROBLEM	(2102, X'836')	Insufficient system resources available.
MQRC_SECOND_MARK_NOT_ALLOWED	(2062, X'80E')	A message is already marked.
MQRC_SIGNAL_OUTSTANDING	(2069, X'815')	Signal outstanding for this handle.
MQRC_SIGNAL1_ERROR	(2099, X'833')	Signal field not valid.
MQRC_STORAGE_MEDIUM_FULL	(2192, X'890')	External storage medium is full.
MQRC_STORAGE_NOT_AVAILABLE	(2071, X'817')	Insufficient storage available.
MQRC_SUPPRESSED_BY_EXIT	(2109, X'83D')	Call suppressed by exit program.

MQRC_SYNCPOINT_LIMIT_REACHED

(2024, X'7E8') No more messages can be handled within current unit of work.

MQRC_SYNCPOINT_NOT_AVAILABLE

(2072, X'818') Syncpoint support not available.

MQRC_UNEXPECTED_ERROR

(2195, X'893') Unexpected error occurred.

MQRC_UOW_ENLISTMENT_ERROR

(2354, X'932') Enlistment in global unit of work failed.

MQRC_UOW_MIX_NOT_SUPPORTED

(2355, X'933') Mixture of unit-of-work calls not supported.

MQRC_UOW_NOT_AVAILABLE

(2255, X'8CF') Unit of work not available for the queue manager to use.

MQRC_WAIT_INTERVAL_ERROR

(2090, X'82A') Wait interval in MQGMO not valid.

MQRC_WRONG_GMO_VERSION

(2256, X'8D0') Wrong version of MQGMO supplied.

MQRC_WRONG_MD_VERSION

(2257, X'8D1') Wrong version of MQMD supplied.

For more information on these reason codes, see Appendix A, "Return codes", on page 527.

Usage notes

1. The message retrieved is normally deleted from the queue. This deletion can occur as part of the MQGET call itself, or as part of a syncpoint.
Message deletion does not occur if a browse option is specified on the *GetMsgOpts* parameter (see the *Options* field described in Chapter 8, "MQGMO – Get-message options", on page 95). The browse options are: MQGMO_BROWSE_FIRST, MQGMO_BROWSE_NEXT, and MQGMO_BROWSE_MSG_UNDER_CURSOR.

2. If the MQGMO_LOCK option is specified with one of the browse options, the browsed message is locked so that it is visible only to this handle.
If the MQGMO_UNLOCK option is specified, a previously-locked message is unlocked. No message is retrieved in this case, and the *MsgDesc*, *BufferLength*, *Buffer* and *DataLength* parameters are not checked or altered.

3. If the application issuing the MQGET call is running as an MQ client, it is possible for the message retrieved to be lost if during the processing of the MQGET call the MQ client terminates abnormally or the client connection is severed. This arises because the surrogate that is running on the queue-manager's platform and which issues the MQGET call on the client's behalf cannot detect the loss of the client until the surrogate is about to return the message to the client; this is *after* the message has been removed from the queue. This can occur for both persistent messages and nonpersistent messages.

The risk of losing messages in this way can be eliminated by always retrieving messages within units of work (that is, by specifying the MQGMO_SYNCPOINT option on the MQGET call, and using the MQCMIT or MQBACK calls to commit or back out the unit of work when processing of the message is complete). If MQGMO_SYNCPOINT is specified, and the client terminates abnormally or the connection is severed, the surrogate backs out the unit of work on the queue manager and the message is reinstated on the queue.

MQGET – Usage notes

In principle, the same situation can arise with applications that are running on the queue-manager's platform, but in this case the window during which a message can be lost is very small. However, as with MQ clients the risk can be eliminated by retrieving the message within a unit of work.

4. If an application puts a sequence of messages on a particular queue within a single unit of work, and then commits that unit of work successfully, the messages become available for retrieval as follows:
 - If the queue is a *nonshared* queue (that is, a local queue), all messages within the unit of work become available at the same time.
 - If the queue is a *shared* queue, messages within the unit of work become available in the order in which they were put, but not all at the same time. When the system is heavily laden, it is possible for the first message in the unit of work to be retrieved successfully, but for the MQGET call for the second or subsequent message in the unit of work to fail with `MQRC_NO_MSG_AVAILABLE`. If this occurs, the application should wait a short while and then retry the operation.
5. If an application puts a sequence of messages on the same queue without using message groups, the order of those messages is preserved provided that certain conditions are satisfied. See the usage notes in the description of the MQPUT call for details. If the conditions are satisfied, the messages will be presented to the receiving application in the order in which they were sent, provided that:
 - Only one receiver is getting messages from the queue.

If there are two or more applications getting messages from the queue, they must agree with the sender the mechanism to be used to identify messages that belong to a sequence. For example, the sender could set all of the *CorrelId* fields in the messages in a sequence to a value that was unique to that sequence of messages.
 - The receiver does not deliberately change the order of retrieval, for example by specifying a particular *MsgId* or *CorrelId*.

If the sending application put the messages as a message group, the messages will be presented to the receiving application in the correct order provided that the receiving application specifies the `MQGMO_LOGICAL_ORDER` option on the MQGET call. For more information about message groups, see:

- *MsgFlags* field in MQMD
 - `MQPMO_LOGICAL_ORDER` option in MQPMO
 - `MQGMO_LOGICAL_ORDER` option in MQGMO
6. Applications should test for the feedback code `MQFB_QUIT` in the *Feedback* field of the *MsgDesc* parameter. If this value is found, the application should end. See the *Feedback* field described in Chapter 10, "MQMD – Message descriptor", on page 141 for more information.
 7. If the queue identified by *Hobj* was opened with the `MQOO_SAVE_ALL_CONTEXT` option, and the completion code from the MQGET call is `MQCC_OK` or `MQCC_WARNING`, the context associated with the queue handle *Hobj* is set to the context of the message that has been retrieved (unless the `MQGMO_BROWSE_FIRST`, `MQGMO_BROWSE_NEXT`, or `MQGMO_BROWSE_MSG_UNDER_CURSOR` option is set, in which case the context is marked as not available).

The saved context can be used on a subsequent MQPUT or MQPUT1 call by specifying the `MQPMO_PASS_IDENTITY_CONTEXT` or `MQPMO_PASS_ALL_CONTEXT` options. This enables the context of the message received to be transferred in whole or in part to another message (for

example, when the message is forwarded to another queue). For more information on message context, see the *WebSphere MQ Application Programming Guide*.

8. If the MQGMO_CONVERT option is included in the *GetMsgOpts* parameter, the application message data is converted to the representation requested by the receiving application, before the data is placed in the *Buffer* parameter:
 - The *Format* field in the control information in the message identifies the structure of the application data, and the *CodedCharSetId* and *Encoding* fields in the control information in the message specify its character-set identifier and encoding.
 - The application issuing the MQGET call specifies in the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter the character-set identifier and encoding to which the application message data should be converted.

When conversion of the message data is necessary, the conversion is performed either by the queue manager itself or by a user-written exit, depending on the value of the *Format* field in the control information in the message:

- The format names listed below are formats that are converted by the queue manager; these are called “built-in” formats:
 - MQFMT_ADMIN
 - MQFMT_CICS (z/OS only)
 - MQFMT_COMMAND_1
 - MQFMT_COMMAND_2
 - MQFMT_DEAD_LETTER_HEADER
 - MQFMT_DIST_HEADER
 - MQFMT_EVENT version 1
 - MQFMT_EVENT version 2 (z/OS only)
 - MQFMT_IMS
 - MQFMT_IMS_VAR_STRING
 - MQFMT_MD_EXTENSION
 - MQFMT_PCF
 - MQFMT_REF_MSG_HEADER
 - MQFMT_RF_HEADER
 - MQFMT_RF_HEADER_2
 - MQFMT_STRING
 - MQFMT_TRIGGER
 - MQFMT_WORK_INFO_HEADER (z/OS only)
 - MQFMT_XMIT_Q_HEADER
- The format name MQFMT_NONE is a special value that indicates that the nature of the data in the message is undefined. As a consequence, the queue manager does not attempt conversion when the message is retrieved from the queue.

Note: If MQGMO_CONVERT is specified on the MQGET call for a message that has a format name of MQFMT_NONE, and the character set or encoding of the message differs from that specified in the *MsgDesc* parameter, the message is still returned in the *Buffer* parameter (assuming no other errors), but the call completes with completion code MQCC_WARNING and reason code MQRC_FORMAT_ERROR.

MQFMT_NONE can be used either when the nature of the message data means that it does not require conversion, or when the sending and receiving applications have agreed between themselves the form in which the message data should be sent.

MQGET – Usage notes

- All other format names cause the message to be passed to a user-written exit for conversion. The exit has the same name as the format, apart from environment-specific additions. User-specified format names should not begin with the letters “MQ”, as such names may conflict with format names supported in the future.

See Appendix F, “Data conversion”, on page 581 for details of the data-conversion exit.

User data in the message can be converted between any supported character sets and encodings. However, be aware that if the message contains one or more MQ header structures, the message cannot be converted from or to a character set that has double-byte or multi-byte characters for any of the characters that are valid in queue names. Reason code MQRC_SOURCE_CCSID_ERROR or MQRC_TARGET_CCSID_ERROR results if this is attempted, and the message is returned unconverted. Unicode character set UCS-2 is an example of such a character set.

On return from MQGET, the following reason code indicates that the message was converted successfully:

MQRC_NONE

The following reason code indicates that the message *may* have been converted successfully; the application should check the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter to find out:

MQRC_TRUNCATED_MSG_ACCEPTED

All other reason codes indicate that the message was not converted.

Note: The interpretation of the reason code described above will be true for conversions performed by user-written exits *only* if the exit conforms to the processing guidelines described in Appendix F, “Data conversion”, on page 581.

9. For the built-in formats listed above, the queue manager may perform *default conversion* of character strings in the message when the MQGMO_CONVERT option is specified. Default conversion allows the queue manager to use an installation-specified default character set that approximates the actual character set, when converting string data. As a result, the MQGET call can succeed with completion code MQCC_OK, instead of completing with MQCC_WARNING and reason code MQRC_SOURCE_CCSID_ERROR or MQRC_TARGET_CCSID_ERROR.

Note: The result of using an approximate character set to convert string data is that some characters may be converted incorrectly. This can be avoided by using in the string only characters which are common to both the actual character set and the default character set.

Default conversion applies both to the application message data and to character fields in the MQMD and MQMDE structures:

- Default conversion of the application message data occurs only when *all* of the following are true:
 - The application specifies MQGMO_CONVERT.
 - The message contains data that must be converted either from or to a character set which is not supported.
 - Default conversion was enabled when the queue manager was installed or restarted.

- Default conversion of the character fields in the MQMD and MQMDE structures occurs as necessary, provided that default conversion is enabled for the queue manager. The conversion is performed even if the MQGMO_CONVERT option is not specified by the application on the MQGET call.
10. For the Visual Basic programming language, the following points apply:
- If the size of the *Buffer* parameter is less than the length specified by the *BufferLength* parameter, the call fails with reason code MQRC_STORAGE_NOT_AVAILABLE.
 - The *Buffer* parameter is declared as being of type String. If the data to be retrieved from the queue is not of type String, the MQGETAny call should be used in place of MQGET.
- The MQGETAny call has the same parameters as the MQGET call, except that the *Buffer* parameter is declared as being of type Any, allowing any type of data to be retrieved. However, this means that *Buffer* cannot be checked to ensure that it is at least *BufferLength* bytes in size.
11. On Compaq NonStop Kernel, the following restrictions apply:
- If the MQGET call is issued outside a Compaq TMF transaction *without* the MQGMO_NO_SYNCPOINT option, the reason code MQRC_UNIT_OF_WORK_NOT_STARTED is returned.
 - If the MQGMO_CONVERT option is specified for an MQGET call, and the message that is retrieved is not in one of the built-in formats (MQFMT_*), the message is passed to the data-conversion exit for conversion.
- Only a single data-conversion exit can be supported by MQSeries because the Compaq NonStop Kernel operating system does not support dynamic linking. The format name of the unconverted message (from the MQMD of the message) is passed to the data-conversion exit in the *MsgDesc* parameter of the exit.

Language invocations

The MQGET call is supported in the programming languages shown below.

C invocation

```
MQGET (Hconn, Hobj, &MsgDesc, &GetMsgOpts, BufferLength, Buffer,
      &DataLength, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;          /* Connection handle */
MQHOBJ   Hobj;           /* Object handle */
MQMD     MsgDesc;       /* Message descriptor */
MQGMO    GetMsgOpts;    /* Options that control the action of MQGET */
MQLONG   BufferLength;   /* Length in bytes of the Buffer area */
MQBYTE   Buffer[n];     /* Area to contain the message data */
MQLONG   DataLength;    /* Length of the message */
MQLONG   CompCode;     /* Completion code */
MQLONG   Reason;       /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQGET' USING HCONN, HOBJ, MSGDESC, GETMSGOPTS, BUFFERLENGTH,
                  BUFFER, DATALENGTH, COMPCODE, REASON.
```

Declare the parameters as follows:

MQGET – Language invocations

```
** Connection handle
01 HCONN      PIC S9(9) BINARY.
** Object handle
01 HOBJ       PIC S9(9) BINARY.
** Message descriptor
01 MSGDESC.
   COPY CMQMDV.
** Options that control the action of MQGET
01 GETMSGOPTS.
   COPY CMQGMV.
** Length in bytes of the BUFFER area
01 BUFFERLENGTH PIC S9(9) BINARY.
** Area to contain the message data
01 BUFFER      PIC X(n).
** Length of the message
01 DATALENGTH PIC S9(9) BINARY.
** Completion code
01 COMPCODE    PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON      PIC S9(9) BINARY.
```

PL/I invocation

```
call MQGET (Hconn, Hobj, MsgDesc, GetMsgOpts, BufferLength, Buffer,
            DataLength, CompCode, Reason);
```

Declare the parameters as follows:

```
dc1 Hconn      fixed bin(31); /* Connection handle */
dc1 Hobj       fixed bin(31); /* Object handle */
dc1 MsgDesc    like MQMD;    /* Message descriptor */
dc1 GetMsgOpts like MQGMO;    /* Options that control the action of
                               MQGET */
dc1 BufferLength fixed bin(31); /* Length in bytes of the Buffer
                               area */
dc1 Buffer      char(n);      /* Area to contain the message data */
dc1 DataLength fixed bin(31); /* Length of the message */
dc1 CompCode   fixed bin(31); /* Completion code */
dc1 Reason     fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
CALL MQGET,(HCONN,HOBJ,MSGDESC,GETMSGOPTS,BUFFERLENGTH,      X
            BUFFER,DATALENGTH,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN      DS      F      Connection handle
HOBJ       DS      F      Object handle
MSGDESC    CMQMDA  ,      Message descriptor
GETMSGOPTS CMQGMOA ,      Options that control the action of MQGET
BUFFERLENGTH DS     F      Length in bytes of the BUFFER area
BUFFER     DS     CL(n)  Area to contain the message data
DATALENGTH DS     F      Length of the message
COMPCODE   DS     F      Completion code
REASON     DS     F      Reason code qualifying COMPCODE
```

TAL invocation

```
INT(32) .EXT Hconn;
INT(32) .EXT Hobj;
STRUCT  .EXT MsgDesc(MQMD^Def);
STRUCT  .EXT GetMsgOpt(MQGMO^Def);
INT(32) .EXT BufferLen;
INT(32) .EXT Buffer[0:BUFFER^LEN];
INT(32) .EXT CC;
INT(32) .EXT Reason;
```

MQGET – Language invocations

```
CALL MQGET(HConn, HObj, MsgDesc, GetMsgOpt, BufferLen, Buffer,  
           DataLen, CC, Reason);
```

Visual Basic invocation

```
MQGET Hconn, Hobj, MsgDesc, GetMsgOpts, BufferLength, Buffer,  
      DataLength, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn      As Long   'Connection handle'  
Dim Hobj       As Long   'Object handle'  
Dim MsgDesc    As MQMD   'Message descriptor'  
Dim GetMsgOpts As MQGMO  'Options that control the action of MQGET'  
Dim BufferLength As Long  'Length in bytes of the Buffer area'  
Dim Buffer      As String 'Area to contain the message data'  
Dim DataLength As Long   'Length of the message'  
Dim CompCode   As Long   'Completion code'  
Dim Reason     As Long   'Reason code qualifying CompCode'
```

MQGET – Language invocations

Chapter 35. MQINQ – Inquire object attributes

The MQINQ call returns an array of integers and a set of character strings containing the attributes of an object. The following types of object are valid:

- Queue
- Namelist
- Process definition
- Queue manager

Namelists are supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, Linux, OS/400, Solaris, Windows, plus WebSphere MQ clients connected to these systems.

Process definitions are not supported in the following environments: Windows 3.1, Windows 95, Windows 98, and VSE/ESA.

Syntax

MQINQ (*Hconn*, *Hobj*, *SelectorCount*, *Selectors*, *IntAttrCount*,
IntAttrs, *CharAttrLength*, *CharAttrs*, *CompCode*, *Reason*)

Parameters

The MQINQ call has the following parameters.

Hconn (MQHCONN) – input

Connection handle.

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on OS/400 for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Hobj (MQHOBJ) – input

Object handle.

This handle represents the object (of any type) whose attributes are required. The handle must have been returned by a previous MQOPEN call that specified the MQOO_INQUIRE option.

SelectorCount (MQLONG) – input

Count of selectors.

This is the count of selectors that are supplied in the *Selectors* array. It is the number of attributes that are to be returned. Zero is a valid value. The maximum number allowed is 256.

MQINQ – Selectors parameter

Selectors (MQLONG×SelectorCount) – input

Array of attribute selectors.

This is an array of *SelectorCount* attribute selectors; each selector identifies an attribute (integer or character) whose value is required.

Each selector must be valid for the type of object that *Hobj* represents, otherwise the call fails with completion code MQCC_FAILED and reason code MQRC_SELECTOR_ERROR.

In the special case of queues:

- If the selector is not valid for queues of *any* type, the call fails with completion code MQCC_FAILED and reason code MQRC_SELECTOR_ERROR.
- If the selector is applicable *only* to queues of type or types other than that of the object, the call succeeds with completion code MQCC_WARNING and reason code MQRC_SELECTOR_NOT_FOR_TYPE.
- If the queue being inquired is a cluster queue, the selectors that are valid depend on how the queue was resolved; see usage note 4 for further details.

Selectors can be specified in any order. Attribute values that correspond to integer attribute selectors (MQIA_* selectors) are returned in *IntAttrs* in the same order in which these selectors occur in *Selectors*. Attribute values that correspond to character attribute selectors (MQCA_* selectors) are returned in *CharAttrs* in the same order in which those selectors occur. MQIA_* selectors can be interleaved with the MQCA_* selectors; only the relative order within each type is important.

Notes:

1. The integer and character attribute selectors are allocated within two different ranges; the MQIA_* selectors reside within the range MQIA_FIRST through MQIA_LAST, and the MQCA_* selectors within the range MQCA_FIRST through MQCA_LAST.
For each range, the constants MQIA_LAST_USED and MQCA_LAST_USED define the highest value that the queue manager will accept.
2. If all of the MQIA_* selectors occur first, the same element numbers can be used to address corresponding elements in the *Selectors* and *IntAttrs* arrays.
3. If the *SelectorCount* parameter is zero, *Selectors* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler may be null.

The attributes that can be inquired are listed in the following tables. For the MQCA_* selectors, the constant that defines the length in bytes of the resulting string in *CharAttrs* is given in parentheses.

Table 86. MQINQ attribute selectors for queues. See the bottom of the table for an explanation of the notes.

Selector	Description	Note
MQCA_ALTERATION_DATE	Date of most-recent alteration (MQ_DATE_LENGTH).	1
MQCA_ALTERATION_TIME	Time of most-recent alteration (MQ_TIME_LENGTH).	1
MQCA_BACKOUT_REQ_Q_NAME	Excessive backout requeue name (MQ_Q_NAME_LENGTH).	5
MQCA_BASE_Q_NAME	Name of queue that alias resolves to (MQ_Q_NAME_LENGTH).	–
MQCA_CF_STRUC_NAME	Coupling-facility structure name (MQ_CF_STRUC_NAME_LENGTH).	3
MQCA_CLUSTER_NAME	Cluster name (MQ_CLUSTER_NAME_LENGTH).	1

Table 86. MQINQ attribute selectors for queues (continued). See the bottom of the table for an explanation of the notes.

Selector	Description	Note
MQCA_CLUSTER_NAMELIST	Cluster namelist (MQ_NAMELIST_NAME_LENGTH).	1
MQCA_CREATION_DATE	Queue creation date (MQ_CREATION_DATE_LENGTH).	–
MQCA_CREATION_TIME	Queue creation time (MQ_CREATION_TIME_LENGTH).	–
MQCA_INITIATION_Q_NAME	Initiation queue name (MQ_Q_NAME_LENGTH).	–
MQCA_PROCESS_NAME	Name of process definition (MQ_PROCESS_NAME_LENGTH).	–
MQCA_Q_DESC	Queue description (MQ_Q_DESC_LENGTH).	–
MQCA_Q_NAME	Queue name (MQ_Q_NAME_LENGTH).	–
MQCA_REMOTE_Q_MGR_NAME	Name of remote queue manager (MQ_Q_MGR_NAME_LENGTH).	–
MQCA_REMOTE_Q_NAME	Name of remote queue as known on remote queue manager (MQ_Q_NAME_LENGTH).	–
MQCA_STORAGE_CLASS	Name of storage class (MQ_STORAGE_CLASS_LENGTH).	3
MQCA_TRIGGER_DATA	Trigger data (MQ_TRIGGER_DATA_LENGTH).	5
MQCA_XMIT_Q_NAME	Transmission queue name (MQ_Q_NAME_LENGTH).	–
MQIA_BACKOUT_THRESHOLD	Backout threshold.	5
MQIA_CURRENT_Q_DEPTH	Number of messages on queue.	–
MQIA_DEF_BIND	Default binding.	1
MQIA_DEF_INPUT_OPEN_OPTION	Default open-for-input option.	5
MQIA_DEF_PERSISTENCE	Default message persistence.	–
MQIA_DEF_PRIORITY	Default message priority.	5
MQIA_DEFINITION_TYPE	Queue definition type.	–
MQIA_DIST_LISTS	Distribution list support.	2
MQIA_HARDEN_GET_BACKOUT	Whether to harden backout count.	5
MQIA_INDEX_TYPE	Type of index maintained for queue.	3
MQIA_INHIBIT_GET	Whether get operations are allowed.	–
MQIA_INHIBIT_PUT	Whether put operations are allowed.	–
MQIA_MAX_MSG_LENGTH	Maximum message length.	–
MQIA_MAX_Q_DEPTH	Maximum number of messages allowed on queue.	–
MQIA_MSG_DELIVERY_SEQUENCE	Whether message priority is relevant.	5
MQIA_OPEN_INPUT_COUNT	Number of MQOPEN calls that have the queue open for input.	–
MQIA_OPEN_OUTPUT_COUNT	Number of MQOPEN calls that have the queue open for output.	–
MQIA_Q_DEPTH_HIGH_EVENT	Control attribute for queue depth high events.	4, 5
MQIA_Q_DEPTH_HIGH_LIMIT	High limit for queue depth.	4, 5
MQIA_Q_DEPTH_LOW_EVENT	Control attribute for queue depth low events.	4, 5
MQIA_Q_DEPTH_LOW_LIMIT	Low limit for queue depth.	4, 5
MQIA_Q_DEPTH_MAX_EVENT	Control attribute for queue depth max events.	4, 5
MQIA_Q_SERVICE_INTERVAL	Limit for queue service interval.	4, 5
MQIA_Q_SERVICE_INTERVAL_EVENT	Control attribute for queue service interval events.	4, 5
MQIA_Q_TYPE	Queue type.	–
MQIA_QSG_DISP	Queue-sharing group disposition.	3
MQIA_RETENTION_INTERVAL	Queue retention interval.	5
MQIA_SCOPE	Queue definition scope.	4, 5
MQIA_SHAREABILITY	Whether queue can be shared for input.	–
MQIA_TRIGGER_CONTROL	Trigger control.	–
MQIA_TRIGGER_DEPTH	Trigger depth.	5
MQIA_TRIGGER_MSG_PRIORITY	Threshold message priority for triggers.	5
MQIA_TRIGGER_TYPE	Trigger type.	–
MQIA_USAGE	Usage.	–

MQINQ – Selectors parameter

Table 86. MQINQ attribute selectors for queues (continued). See the bottom of the table for an explanation of the notes.

Selector	Description	Note
Notes:		
1. Supported on AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.		
2. Supported on AIX, HP-UX, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.		
3. Supported on z/OS.		
4. Not supported on z/OS.		
5. Not supported on VSE/ESA.		

Table 87. MQINQ attribute selectors for namelists. See the bottom of Table 86 on page 394 for an explanation of the notes.

Selector	Description	Note
MQCA_ALTERATION_DATE	Date of most-recent alteration (MQ_DATE_LENGTH).	1
MQCA_ALTERATION_TIME	Time of most-recent alteration (MQ_TIME_LENGTH).	1
MQCA_NAMELIST_DESC	Namelist description (MQ_NAMELIST_DESC_LENGTH).	1
MQCA_NAMELIST_NAME	Name of namelist object (MQ_NAMELIST_NAME_LENGTH).	1
MQIA_NAMELIST_TYPE	Namelist type.	3
MQCA_NAMES	Names in the namelist (MQ_Q_NAME_LENGTH × Number of names in the list).	1
MQIA_NAME_COUNT	Number of names in the namelist.	1
MQIA_QSG_DISP	Queue-sharing group disposition.	3

Table 88. MQINQ attribute selectors for process definitions. See the bottom of Table 86 on page 394 for an explanation of the notes.

Selector	Description	Note
MQCA_ALTERATION_DATE	Date of most-recent alteration (MQ_DATE_LENGTH).	1
MQCA_ALTERATION_TIME	Time of most-recent alteration (MQ_TIME_LENGTH).	1
MQCA_APPL_ID	Application identifier (MQ_PROCESS_APPL_ID_LENGTH).	5
MQCA_ENV_DATA	Environment data (MQ_PROCESS_ENV_DATA_LENGTH).	5
MQCA_PROCESS_DESC	Description of process definition (MQ_PROCESS_DESC_LENGTH).	5
MQCA_PROCESS_NAME	Name of process definition (MQ_PROCESS_NAME_LENGTH).	5
MQCA_USER_DATA	User data (MQ_PROCESS_USER_DATA_LENGTH).	5
MQIA_APPL_TYPE	Application type.	5
MQIA_QSG_DISP	Queue-sharing group disposition.	3

Table 89. MQINQ attribute selectors for the queue manager. See the bottom of Table 86 on page 394 for an explanation of the notes.

Selector	Description	Note
MQCA_ALTERATION_DATE	Date of most-recent alteration (MQ_DATE_LENGTH).	1
MQCA_ALTERATION_TIME	Time of most-recent alteration (MQ_TIME_LENGTH).	1
MQCA_CHANNEL_AUTO_DEF_EXIT	Automatic channel definition exit name (MQ_EXIT_NAME_LENGTH).	1
MQCA_CLUSTER_WORKLOAD_DATA	Data passed to cluster workload exit (MQ_EXIT_DATA_LENGTH).	1
MQCA_CLUSTER_WORKLOAD_EXIT	Name of cluster workload exit (MQ_EXIT_NAME_LENGTH).	1
MQCA_COMMAND_INPUT_Q_NAME	System command input queue name (MQ_Q_NAME_LENGTH).	5
MQCA_DEAD_LETTER_Q_NAME	Name of dead-letter queue (MQ_Q_NAME_LENGTH).	5
MQCA_DEF_XMIT_Q_NAME	Default transmission queue name (MQ_Q_NAME_LENGTH).	5
MQCA_IGQ_USER_ID	Intra-group queuing user identifier (MQ_USER_ID_LENGTH).	3
MQCA_Q_MGR_DESC	Queue manager description (MQ_Q_MGR_DESC_LENGTH).	5

Table 89. MQINQ attribute selectors for the queue manager (continued). See the bottom of Table 86 on page 394 for an explanation of the notes.

Selector	Description	Note
MQCA_Q_MGR_IDENTIFIER	Queue-manager identifier (MQ_Q_MGR_IDENTIFIER_LENGTH).	1
MQCA_Q_MGR_NAME	Name of local queue manager (MQ_Q_MGR_NAME_LENGTH).	5
MQCA_QSG_NAME	Queue-sharing group name (MQ_QSG_NAME_LENGTH).	3
MQCA_REPOSITORY_NAME	Name of cluster for which queue manager provides repository services (MQ_Q_MGR_NAME_LENGTH).	1
MQCA_REPOSITORY_NAMELIST	Name of namelist object containing names of clusters for which queue manager provides repository services (MQ_NAMELIST_NAME_LENGTH).	1
MQIA_AUTHORITY_EVENT	Control attribute for authority events.	4, 5
MQIA_CHANNEL_AUTO_DEF	Control attribute for automatic channel definition.	2
MQIA_CHANNEL_AUTO_DEF_EVENT	Control attribute for automatic channel definition events.	2
MQIA_CLUSTER_WORKLOAD_LENGTH	Cluster workload length.	1
MQIA_CODED_CHAR_SET_ID	Coded character set identifier.	5
MQIA_COMMAND_LEVEL	Command level supported by queue manager.	5
MQIA_DIST_LISTS	Distribution list support.	2
MQIA_EXPIRY_INTERVAL	Interval between scans for expired messages.	3
MQIA_IGQ_PUT_AUTHORITY	Intra-group queuing put authority.	3
MQIA_INHIBIT_EVENT	Control attribute for inhibit events.	4, 5
MQIA_INTRA_GROUP_QUEUING	Intra-group queuing support.	3
MQIA_LOCAL_EVENT	Control attribute for local events.	4, 5
MQIA_MAX_HANDLES	Maximum number of handles.	5
MQIA_MAX_MSG_LENGTH	Maximum message length.	5
MQIA_MAX_PRIORITY	Maximum priority.	5
MQIA_MAX_UNCOMMITTED_MSGS	Maximum number of uncommitted messages within a unit of work.	5
MQIA_PERFORMANCE_EVENT	Control attribute for performance events.	4, 5
MQIA_PLATFORM	Platform on which the queue manager resides.	5
MQIA_REMOTE_EVENT	Control attribute for remote events.	4, 5
MQIA_START_STOP_EVENT	Control attribute for start stop events.	4, 5
MQIA_SYNCPOINT	Syncpoint availability.	5
MQIA_TRIGGER_INTERVAL	Trigger interval.	5

IntAttrCount (MQLONG) – input

Count of integer attributes.

This is the number of elements in the *IntAttrs* array. Zero is a valid value.

If this is at least the number of MQIA_* selectors in the *Selectors* parameter, all integer attributes requested are returned.

IntAttrs (MQLONG×IntAttrCount) – output

Array of integer attributes.

This is an array of *IntAttrCount* integer attribute values.

Integer attribute values are returned in the same order as the MQIA_* selectors in the *Selectors* parameter. If the array contains more elements than the number of MQIA_* selectors, the excess elements are unchanged.

MQINQ – IntAttrs parameter

If *Hobj* represents a queue, but an attribute selector is not applicable to that type of queue, the specific value MQIAV_NOT_APPLICABLE is returned for the corresponding element in the *IntAttrs* array.

If the *IntAttrCount* or *SelectorCount* parameter is zero, *IntAttrs* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler may be null.

CharAttrLength (MQLONG) – input

Length of character attributes buffer.

This is the length in bytes of the *CharAttrs* parameter.

This must be at least the sum of the lengths of the requested character attributes (see *Selectors*). Zero is a valid value.

CharAttrs (MQCHAR×CharAttrLength) – output

Character attributes.

This is the buffer in which the character attributes are returned, concatenated together. The length of the buffer is given by the *CharAttrLength* parameter.

Character attributes are returned in the same order as the MQCA_* selectors in the *Selectors* parameter. The length of each attribute string is fixed for each attribute (see *Selectors*), and the value in it is padded to the right with blanks if necessary. If the buffer is larger than that needed to contain all of the requested character attributes (including padding), the bytes beyond the last attribute value returned are unchanged.

If *Hobj* represents a queue, but an attribute selector is not applicable to that type of queue, a character string consisting entirely of asterisks (*) is returned as the value of that attribute in *CharAttrs*.

If the *CharAttrLength* or *SelectorCount* parameter is zero, *CharAttrs* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler may be null.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_WARNING

Warning (partial completion).

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

- MQRC_CHAR_ATTRS_TOO_SHORT**
(2008, X'7D8') Not enough space allowed for character attributes.
- MQRC_INT_ATTR_COUNT_TOO_SMALL**
(2022, X'7E6') Not enough space allowed for integer attributes.
- MQRC_SELECTOR_NOT_FOR_TYPE**
(2068, X'814') Selector not applicable to queue type.

If *CompCode* is MQCC_FAILED:

- MQRC_ADAPTER_NOT_AVAILABLE**
(2204, X'89C') Adapter not available.
- MQRC_ADAPTER_SERV_LOAD_ERROR**
(2130, X'852') Unable to load adapter service module.
- MQRC_API_EXIT_ERROR**
(2374, X'946') API exit failed.
- MQRC_API_EXIT_LOAD_ERROR**
(2183, X'887') Unable to load API exit.
- MQRC_ASID_MISMATCH**
(2157, X'86D') Primary and home ASIDs differ.
- MQRC_CALL_IN_PROGRESS**
(2219, X'8AB') MQI call entered before previous call complete.
- MQRC_CF_STRUC_FAILED**
(2373, X'945') Coupling-facility structure failed.
- MQRC_CF_STRUC_IN_USE**
(2346, X'92A') Coupling-facility structure in use.
- MQRC_CHAR_ATTR_LENGTH_ERROR**
(2006, X'7D6') Length of character attributes not valid.
- MQRC_CHAR_ATTRS_ERROR**
(2007, X'7D7') Character attributes string not valid.
- MQRC_CICS_WAIT_FAILED**
(2140, X'85C') Wait request rejected by CICS.
- MQRC_CONNECTION_BROKEN**
(2009, X'7D9') Connection to queue manager lost.
- MQRC_CONNECTION_NOT_AUTHORIZED**
(2217, X'8A9') Not authorized for connection.
- MQRC_CONNECTION_STOPPING**
(2203, X'89B') Connection shutting down.
- MQRC_HCONN_ERROR**
(2018, X'7E2') Connection handle not valid.
- MQRC_HOBJ_ERROR**
(2019, X'7E3') Object handle not valid.
- MQRC_INT_ATTR_COUNT_ERROR**
(2021, X'7E5') Count of integer attributes not valid.
- MQRC_INT_ATTRS_ARRAY_ERROR**
(2023, X'7E7') Integer attributes array not valid.
- MQRC_NOT_OPEN_FOR_INQUIRE**
(2038, X'7F6') Queue not open for inquire.
- MQRC_OBJECT_CHANGED**
(2041, X'7F9') Object definition changed since opened.
- MQRC_OBJECT_DAMAGED**
(2101, X'835') Object damaged.
- MQRC_PAGESET_ERROR**
(2193, X'891') Error accessing page-set data set.
- MQRC_Q_DELETED**
(2052, X'804') Queue has been deleted.

MQINQ – Reason parameter

MQRC_Q_MGR_NAME_ERROR	(2058, X'80A')	Queue manager name not valid or not known.
MQRC_Q_MGR_NOT_AVAILABLE	(2059, X'80B')	Queue manager not available for connection.
MQRC_Q_MGR_STOPPING	(2162, X'872')	Queue manager shutting down.
MQRC_RESOURCE_PROBLEM	(2102, X'836')	Insufficient system resources available.
MQRC_SELECTOR_COUNT_ERROR	(2065, X'811')	Count of selectors not valid.
MQRC_SELECTOR_ERROR	(2067, X'813')	Attribute selector not valid.
MQRC_SELECTOR_LIMIT_EXCEEDED	(2066, X'812')	Count of selectors too big.
MQRC_STORAGE_NOT_AVAILABLE	(2071, X'817')	Insufficient storage available.
MQRC_SUPPRESSED_BY_EXIT	(2109, X'83D')	Call suppressed by exit program.
MQRC_UNEXPECTED_ERROR	(2195, X'893')	Unexpected error occurred.

For more information on these reason codes, see Appendix A, “Return codes”, on page 527.

Usage notes

1. The values returned are a snapshot of the selected attributes. There is no guarantee that the attributes will not change before the application can act upon the returned values.
2. When you open a model queue, a dynamic local queue is created. This is true even if you open the model queue to inquire about its attributes.
The attributes of the dynamic queue (with certain exceptions) are the same as those of the model queue at the time the dynamic queue is created. If you subsequently use the MQINQ call on this queue, the queue manager returns the attributes of the dynamic queue, and not those of the model queue. See Table 92 on page 459 for details of which attributes of the model queue are inherited by the dynamic queue.
3. If the object being inquired is an alias queue, the attribute values returned by the MQINQ call are those of the alias queue, and not those of the base queue to which the alias resolves.
4. If the object being inquired is a cluster queue, the attributes that can be inquired depend on how the queue is opened:
 - If the cluster queue is opened for inquire plus one or more of input, browse, or set, there must be a local instance of the cluster queue in order for the open to succeed. In this case the attributes that can be inquired are those valid for local queues.
 - If the cluster queue is opened for inquire alone, or inquire and output, only the attributes listed below can be inquired; the *QType* attribute has the value MQQT_CLUSTER in this case:
 - MQCA_Q_DESC
 - MQCA_Q_NAME
 - MQIA_DEF_BIND
 - MQIA_DEF_PERSISTENCE
 - MQIA_DEF_PRIORITY
 - MQIA_INHIBIT_PUT

MQIA_Q_TYPE

If the cluster queue is opened with no fixed binding (that is, MQOO_BIND_NOT_FIXED specified on the MQOPEN call, or MQOO_BIND_AS_Q_DEF specified when the *DefBind* attribute has the value MQBND_BIND_NOT_FIXED), successive MQINQ calls for the queue may inquire different instances of the cluster queue, although usually all of the instances have the same attribute values.

For more information about cluster queues, refer to the *WebSphere MQ Queue Manager Clusters* book.

5. If a number of attributes are to be inquired, and subsequently some of them are to be set using the MQSET call, it may be convenient to position at the beginning of the selector arrays the attributes that are to be set, so that the same arrays (with reduced counts) can be used for MQSET.
6. If more than one of the warning situations arise (see the *CompCode* parameter), the reason code returned is the *first* one in the following list that applies:
 - a. MQRC_SELECTOR_NOT_FOR_TYPE
 - b. MQRC_INT_ATTR_COUNT_TOO_SMALL
 - c. MQRC_CHAR_ATTRS_TOO_SHORT
7. For more information about object attributes, see:
 - Chapter 40, “Attributes for queues”, on page 457
 - Chapter 41, “Attributes for namelists”, on page 491
 - Chapter 42, “Attributes for process definitions”, on page 495
 - Chapter 43, “Attributes for the queue manager”, on page 501

Language invocations

The MQINQ call is supported in the programming languages shown below.

C invocation

```
MQINQ (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,
      CharAttrLength, CharAttrs, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;          /* Connection handle */
MQHOBJ  Hobj;           /* Object handle */
MQLONG  SelectorCount; /* Count of selectors */
MQLONG  Selectors[n];  /* Array of attribute selectors */
MQLONG  IntAttrCount;  /* Count of integer attributes */
MQLONG  IntAttrs[n];   /* Array of integer attributes */
MQLONG  CharAttrLength; /* Length of character attributes buffer */
MQCHAR  CharAttrs[n];  /* Character attributes */
MQLONG  CompCode;      /* Completion code */
MQLONG  Reason;        /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQINQ' USING HCONN, HOBJ, SELECTORCOUNT, SELECTORS-TABLE,
                  INTATTRCOUNT, INTATTRS-TABLE, CHARATTRLENGTH,
                  CHARATTRS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN          PIC S9(9) BINARY.
** Object handle
01 HOBJ           PIC S9(9) BINARY.
```

MQINQ – Language invocations

```
** Count of selectors
01 SELECTORCOUNT PIC S9(9) BINARY.
** Array of attribute selectors
01 SELECTORS-TABLE.
02 SELECTORS PIC S9(9) BINARY OCCURS n TIMES.
** Count of integer attributes
01 INTATTRCOUNT PIC S9(9) BINARY.
** Array of integer attributes
01 INTATTRS-TABLE.
02 INTATTRS PIC S9(9) BINARY OCCURS n TIMES.
** Length of character attributes buffer
01 CHARATTRLENGTH PIC S9(9) BINARY.
** Character attributes
01 CHARATTRS PIC X(n).
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON PIC S9(9) BINARY.
```

PL/I invocation

```
call MQINQ (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount,
           IntAttrs, CharAttrLength, CharAttrs, CompCode, Reason);
```

Declare the parameters as follows:

```
dc1 Hconn          fixed bin(31); /* Connection handle */
dc1 Hobj           fixed bin(31); /* Object handle */
dc1 SelectorCount  fixed bin(31); /* Count of selectors */
dc1 Selectors(n)   fixed bin(31); /* Array of attribute selectors */
dc1 IntAttrCount   fixed bin(31); /* Count of integer attributes */
dc1 IntAttrs(n)    fixed bin(31); /* Array of integer attributes */
dc1 CharAttrLength fixed bin(31); /* Length of character attributes
                                buffer */
dc1 CharAttrs      char(n);       /* Character attributes */
dc1 CompCode       fixed bin(31); /* Completion code */
dc1 Reason         fixed bin(31); /* Reason code qualifying
                                CompCode */
```

System/390 assembler invocation

```
CALL MQINQ, (HCONN,HOBJ,SELECTORCOUNT,SELECTORS,INTATTRCOUNT, X
           INTATTRS,CHARATTRLENGTH,CHARATTRS,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN          DS F      Connection handle
HOBJ           DS F      Object handle
SELECTORCOUNT DS F      Count of selectors
SELECTORS      DS (n)F   Array of attribute selectors
INTATTRCOUNT  DS F      Count of integer attributes
INTATTRS      DS (n)F   Array of integer attributes
CHARATTRLENGTH DS F      Length of character attributes buffer
CHARATTRS     DS CL(n)  Character attributes
COMPCODE      DS F      Completion code
REASON        DS F      Reason code qualifying COMPCODE
```

TAL invocation

```
INT(32) .EXT HConn ;
INT(32) .EXT Hobj ;
INT(32) SelectorCount;
INT(32) .EXT Selectors[0:NUM^SELECTORS];
INT(32) IntAttrCount;
INT(32) .EXT IntAttrs[0:NUM^INT^ATTR];
INT(32) CharAttrLength;
STRING .EXT CharAttrs[0:LEN^CHAR^ATTR];
```

```
INT(32) .EXT CC;  
INT(32) .EXT Reason;
```

```
PROC MQINQ(HConn, HObj, SelectorCount, Selectors, IntAttrCount,  
IntAttrs, CharAttrLength, CharAttrs, CC, Reason)
```

Visual Basic invocation

```
MQINQ Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,  
CharAttrLength, CharAttrs, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn          As Long  'Connection handle'  
Dim Hobj           As Long  'Object handle'  
Dim SelectorCount As Long  'Count of selectors'  
Dim Selectors      As Long  'Array of attribute selectors'  
Dim IntAttrCount   As Long  'Count of integer attributes'  
Dim IntAttrs       As Long  'Array of integer attributes'  
Dim CharAttrLength As Long  'Length of character attributes buffer'  
Dim CharAttrs      As String 'Character attributes'  
Dim CompCode       As Long  'Completion code'  
Dim Reason         As Long  'Reason code qualifying CompCode'
```

MQINQ – Language invocations

Chapter 36. MQOPEN – Open object

The MQOPEN call establishes access to an object. The following types of object are valid:

- Queue (including distribution lists)
- Namelist
- Process definition
- Queue manager

Namelists are supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Process definitions are not supported in the following environments: Windows 3.1, Windows 95, Windows 98, and VSE/ESA.

Syntax

MQOPEN (*Hconn*, *ObjDesc*, *Options*, *Hobj*, *CompCode*, *Reason*)

Parameters

The MQOPEN call has the following parameters.

Hconn (MQHCONN) – input

Connection handle.

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on OS/400 for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN

Default connection handle.

ObjDesc (MQOD) – input/output

Object descriptor.

This is a structure that identifies the object to be opened; see Chapter 12, “MQOD – Object descriptor”, on page 211 for details.

If the *ObjectName* field in the *ObjDesc* parameter is the name of a model queue, a dynamic local queue is created with the attributes of the model queue; this happens irrespective of the open options specified by the *Options* parameter. Subsequent operations using the *Hobj* returned by the MQOPEN call are performed on the new dynamic queue, and not on the model queue. This is true even for the MQINQ and MQSET calls. The name of the model queue in the *ObjDesc* parameter is replaced with the name of the dynamic queue created. The

MQOPEN – ObjDesc parameter

type of the dynamic queue is determined by the value of the *DefinitionType* attribute of the model queue (see Chapter 40, “Attributes for queues”, on page 457). For information about the close options applicable to dynamic queues, see the description of the MQCLOSE call.

Options (MQLONG) – input

Options that control the action of MQOPEN.

At least one of the following options must be specified:

MQOO_BROWSE
MQOO_INPUT_* (only one of these)
MQOO_INQUIRE
MQOO_OUTPUT
MQOO_SET

See below for details of these options; other options can be specified as required. If more than one option is required, the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

Combinations that are not valid are noted; all other combinations are valid. Only options that are applicable to the type of object specified by *ObjDesc* are allowed (see Table 90 on page 411).

Access options: The following options control the type of operations that can be performed on the object:

MQOO_INPUT_AS_Q_DEF

Open queue to get messages using queue-defined default.

The queue is opened for use with subsequent MQGET calls. The type of access is either shared or exclusive, depending on the value of the *DefInputOpenOption* queue attribute; see Chapter 40, “Attributes for queues”, on page 457 for details.

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues.

This option is not supported on VSE/ESA.

MQOO_INPUT_SHARED

Open queue to get messages with shared access.

The queue is opened for use with subsequent MQGET calls. The call can succeed if the queue is currently open by this or another application with MQOO_INPUT_SHARED, but fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open with MQOO_INPUT_EXCLUSIVE.

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues.

MQOO_INPUT_EXCLUSIVE

Open queue to get messages with exclusive access.

The queue is opened for use with subsequent MQGET calls. The call fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open by this or another application for input of any type (MQOO_INPUT_SHARED or MQOO_INPUT_EXCLUSIVE).

MQOPEN – Options parameter

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues.

The following notes apply to these options:

- Only one of these options can be specified.
- An MQOPEN call with one of these options can succeed even if the *InhibitGet* queue attribute is set to MQQA_GET_INHIBITED (although subsequent MQGET calls will fail while the attribute is set to this value).
- If the queue is defined as not being shareable (that is, the *Shareability* queue attribute has the value MQQA_NOT_SHAREABLE), attempts to open the queue for shared access are treated as attempts to open the queue with exclusive access.
- If an alias queue is opened with one of these options, the test for exclusive use (or for whether another application has exclusive use) is against the base queue to which the alias resolves.
- These options are not valid if *ObjectQMgrName* is the name of a queue manager alias; this is true even if the value of the *RemoteQMgrName* attribute in the local definition of a remote queue used for queue-manager aliasing is the name of the local queue manager.

MQOO_BROWSE

Open queue to browse messages.

The queue is opened for use with subsequent MQGET calls with one of the following options:

MQGMO_BROWSE_FIRST
MQGMO_BROWSE_NEXT
MQGMO_BROWSE_MSG_UNDER_CURSOR

This is allowed even if the queue is currently open for MQOO_INPUT_EXCLUSIVE. An MQOPEN call with the MQOO_BROWSE option establishes a browse cursor, and positions it logically before the first message on the queue; see the *Options* field described in Chapter 8, “MQGMO – Get-message options”, on page 95 for further information.

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects which are not queues. It is also not valid if *ObjectQMgrName* is the name of a queue manager alias; this is true even if the value of the *RemoteQMgrName* attribute in the local definition of a remote queue used for queue-manager aliasing is the name of the local queue manager.

MQOO_OUTPUT

Open queue to put messages.

The queue is opened for use with subsequent MQPUT calls.

An MQOPEN call with this option can succeed even if the *InhibitPut* queue attribute is set to MQQA_PUT_INHIBITED (although subsequent MQPUT calls will fail while the attribute is set to this value).

This option is valid for all types of queue, including distribution lists.

MQOO_INQUIRE

Open object to inquire attributes.

The queue, namelist, process definition, or queue manager is opened for use with subsequent MQINQ calls.

MQOPEN – Options parameter

This option is valid for all types of object other than distribution lists. It is not valid if *ObjectQMGrName* is the name of a queue manager alias; this is true even if the value of the *RemoteQMGrName* attribute in the local definition of a remote queue used for queue-manager aliasing is the name of the local queue manager.

MQOO_SET

Open queue to set attributes.

The queue is opened for use with subsequent MQSET calls.

This option is valid for all types of queue other than distribution lists. It is not valid if *ObjectQMGrName* is the name of a local definition of a remote queue; this is true even if the value of the *RemoteQMGrName* attribute in the local definition of a remote queue used for queue-manager aliasing is the name of the local queue manager.

Binding options: The following options apply when the object being opened is a cluster queue; these options control the binding of the queue handle to a particular instance of the cluster queue:

MQOO_BIND_ON_OPEN

Bind handle to destination when queue is opened.

This causes the local queue manager to bind the queue handle to a particular instance of the destination queue when the queue is opened. As a result, all messages put using this handle are sent to the same instance of the destination queue, and by the same route.

This option is valid only for queues, and affects only cluster queues. If specified for a queue that is not a cluster queue, the option is ignored.

This option is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

MQOO_BIND_NOT_FIXED

Do not bind to a specific destination.

This stops the local queue manager binding the queue handle to a particular instance of the destination queue. As a result, successive MQPUT calls using this handle may result in the messages being sent to *different* instances of the destination queue, or being sent to the same instance but by different routes. It also allows the instance selected to be changed subsequently by the local queue manager, by a remote queue manager, or by a message channel agent (MCA), according to network conditions.

Note: Client and server applications which need to exchange a *series* of messages in order to complete a transaction should not use MQOO_BIND_NOT_FIXED (or MQOO_BIND_AS_Q_DEF when *DefBind* has the value MQBND_BIND_NOT_FIXED), because successive messages in the series may be sent to different instances of the server application.

If MQOO_BROWSE or one of the MQOO_INPUT_* options is specified for a cluster queue, the queue manager is forced to select the local instance of the cluster queue. As a result, the binding of the queue handle is fixed, even if MQOO_BIND_NOT_FIXED is specified.

MQOPEN – Options parameter

If MQOO_INQUIRE is specified with MQOO_BIND_NOT_FIXED, successive MQINQ calls using that handle may inquire different instances of the cluster queue, although usually all of the instances have the same attribute values.

MQOO_BIND_NOT_FIXED is valid only for queues, and affects only cluster queues. If specified for a queue that is not a cluster queue, the option is ignored.

This option is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

MQOO_BIND_AS_Q_DEF

Use default binding for queue.

This causes the local queue manager to bind the queue handle in the way defined by the *DefBind* queue attribute. The value of this attribute is either MQBND_BIND_ON_OPEN or MQBND_BIND_NOT_FIXED.

MQOO_BIND_AS_Q_DEF is the default if neither MQOO_BIND_ON_OPEN nor MQOO_BIND_NOT_FIXED is specified.

MQOO_BIND_AS_Q_DEF is defined to aid program documentation. It is not intended that this option be used with either of the other two bind options, but because its value is zero such use cannot be detected.

This option is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Context options: The following options control the processing of message context:

MQOO_SAVE_ALL_CONTEXT

Save context when message retrieved.

Context information is associated with this queue handle. This information is set from the context of any message retrieved using this handle. For more information on message context, see the *WebSphere MQ Application Programming Guide*.

This context information can be passed to a message that is subsequently put on a queue using the MQPUT or MQPUT1 calls. See the MQPMO_PASS_IDENTITY_CONTEXT and MQPMO_PASS_ALL_CONTEXT options described in Chapter 14, “MQPMO – Put-message options”, on page 229.

Until a message has been successfully retrieved, context cannot be passed to a message being put on a queue.

A message retrieved using one of the MQGMO_BROWSE_* browse options does **not** have its context information saved (although the context fields in the *MsgDesc* parameter are set after a browse).

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects which are not queues. One of the MQOO_INPUT_* options must be specified.

This option is not supported on VSE/ESA.

MQOPEN – Options parameter

MQOO_PASS_IDENTITY_CONTEXT

Allow identity context to be passed.

This allows the MQPMO_PASS_IDENTITY_CONTEXT option to be specified in the *PutMsgOpts* parameter when a message is put on a queue; this gives the message the identity context information from an input queue that was opened with the MQOO_SAVE_ALL_CONTEXT option. For more information on message context, see the *WebSphere MQ Application Programming Guide*.

The MQOO_OUTPUT option must be specified.

This option is valid for all types of queue, including distribution lists.

This option is not supported on VSE/ESA.

MQOO_PASS_ALL_CONTEXT

Allow all context to be passed.

This allows the MQPMO_PASS_ALL_CONTEXT option to be specified in the *PutMsgOpts* parameter when a message is put on a queue; this gives the message the identity and origin context information from an input queue that was opened with the MQOO_SAVE_ALL_CONTEXT option. For more information on message context, see the *WebSphere MQ Application Programming Guide*.

This option implies MQOO_PASS_IDENTITY_CONTEXT, which need not therefore be specified. The MQOO_OUTPUT option must be specified.

This option is valid for all types of queue, including distribution lists.

This option is not supported on VSE/ESA.

MQOO_SET_IDENTITY_CONTEXT

Allow identity context to be set.

This allows the MQPMO_SET_IDENTITY_CONTEXT option to be specified in the *PutMsgOpts* parameter when a message is put on a queue; this gives the message the identity context information contained in the *MsgDesc* parameter specified on the MQPUT or MQPUT1 call. For more information on message context, see the *WebSphere MQ Application Programming Guide*.

This option implies MQOO_PASS_IDENTITY_CONTEXT, which need not therefore be specified. The MQOO_OUTPUT option must be specified.

This option is valid for all types of queue, including distribution lists.

This option is not supported on VSE/ESA.

MQOO_SET_ALL_CONTEXT

Allow all context to be set.

This allows the MQPMO_SET_ALL_CONTEXT option to be specified in the *PutMsgOpts* parameter when a message is put on a queue; this gives the message the identity and origin context information contained in the *MsgDesc* parameter specified on the MQPUT or MQPUT1 call. For more information on message context, see the *WebSphere MQ Application Programming Guide*.

This option implies the following options, which need not therefore be specified:

- MQOO_PASS_IDENTITY_CONTEXT
- MQOO_PASS_ALL_CONTEXT
- MQOO_SET_IDENTITY_CONTEXT

MQOPEN – Options parameter

The MQOO_OUTPUT option must be specified.

This option is valid for all types of queue, including distribution lists.

This option is not supported on VSE/ESA.

Other options: The following options control authorization checking, and what happens when the queue manager is quiescing:

MQOO_ALTERNATE_USER_AUTHORITY

Validate with specified user identifier.

This indicates that the *AlternateUserId* field in the *ObjDesc* parameter contains a user identifier that is to be used to validate this MQOPEN call. The call can succeed only if this *AlternateUserId* is authorized to open the object with the specified access options, regardless of whether the user identifier under which the application is running is authorized to do so. This does not apply to any context options specified, however, which are always checked against the user identifier under which the application is running.

This option is valid for all types of object.

This option is not supported on VSE/ESA.

MQOO_FAIL_IF QUIESCING

Fail if queue manager is quiescing.

This option forces the MQOPEN call to fail if the queue manager is in quiescing state.

On z/OS, for a CICS or IMS application, this option also forces the MQOPEN call to fail if the connection is in quiescing state.

This option is valid for all types of object.

This option is not supported on VSE/ESA. This option is accepted but ignored on: Windows 3.1, Windows 95, Windows 98.

For information about client channels see the *WebSphere MQ Clients* book.

Table 90. Valid MQOPEN options for each queue type

Option	Alias (note 1)	Local and Model	Remote	Nonlocal Cluster	Distribution list
MQOO_INPUT_AS_Q_DEF	Yes	Yes	No	No	No
MQOO_INPUT_SHARED	Yes	Yes	No	No	No
MQOO_INPUT_EXCLUSIVE	Yes	Yes	No	No	No
MQOO_BROWSE	Yes	Yes	No	No	No
MQOO_OUTPUT	Yes	Yes	Yes	Yes	Yes
MQOO_INQUIRE	Yes	Yes	Note 2	Yes	No
MQOO_SET	Yes	Yes	Note 2	No	No
MQOO_BIND_ON_OPEN (note 3)	Yes	Yes	Yes	Yes	Yes
MQOO_BIND_NOT_FIXED (note 3)	Yes	Yes	Yes	Yes	Yes
MQOO_BIND_AS_Q_DEF (note 3)	Yes	Yes	Yes	Yes	Yes
MQOO_SAVE_ALL_CONTEXT	Yes	Yes	No	No	No
MQOO_PASS_IDENTITY_CONTEXT	Yes	Yes	Yes	Yes	Yes
MQOO_PASS_ALL_CONTEXT	Yes	Yes	Yes	Yes	Yes
MQOO_SET_IDENTITY_CONTEXT	Yes	Yes	Yes	Yes	Yes

MQOPEN – Options parameter

Table 90. Valid MQOPEN options for each queue type (continued)

Option	Alias (note 1)	Local and Model	Remote	Nonlocal Cluster	Distribution list
MQOO_SET_ALL_CONTEXT	Yes	Yes	Yes	Yes	Yes
MQOO_ALTERNATE_USER_AUTHORITY	Yes	Yes	Yes	Yes	Yes
MQOO_FAIL_IF QUIESCING	Yes	Yes	Yes	Yes	Yes

Notes:

1. The validity of options for aliases depends on the validity of the option for the queue to which the alias resolves.
2. This option is valid only for the local definition of a remote queue.
3. This option can be specified for any queue type, but is ignored if the queue is not a cluster queue.

Hobj (MQHOBJ) – output

Object handle.

This handle represents the access that has been established to the object. It must be specified on subsequent MQ calls that operate on the object. It ceases to be valid when the MQCLOSE call is issued, or when the unit of processing that defines the scope of the handle terminates.

The scope of the object handle returned is the same as that of the connection handle specified on the call. See the description of the *Hconn* parameter of the MQCONN call for information about handle scope.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_WARNING

Warning (partial completion).

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_MULTIPLE_REASONS

(2136, X'858') Multiple reason codes returned.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE

(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR

(2130, X'852') Unable to load adapter service module.

MQRC_ALIAS_BASE_Q_TYPE_ERROR

(2001, X'7D1') Alias base queue not a valid type.

MQRC_API_EXIT_ERROR
 (2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
 (2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
 (2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
 (2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_NOT_AVAILABLE
 (2345, X'929') Coupling facility not available.

MQRC_CF_STRUC_AUTH_FAILED
 (2348, X'92C') Coupling-facility structure authorization check failed.

MQRC_CF_STRUC_ERROR
 (2349, X'92D') Coupling-facility structure not valid.

MQRC_CF_STRUC_FAILED
 (2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
 (2346, X'92A') Coupling-facility structure in use.

MQRC_CF_STRUC_LIST_HDR_IN_USE
 (2347, X'92B') Coupling-facility structure list-header in use.

MQRC_CICS_WAIT_FAILED
 (2140, X'85C') Wait request rejected by CICS.

MQRC_CLUSTER_EXIT_ERROR
 (2266, X'8DA') Cluster workload exit failed.

MQRC_CLUSTER_PUT_INHIBITED
 (2268, X'8DC') Put calls inhibited for all queues in cluster.

MQRC_CLUSTER_RESOLUTION_ERROR
 (2189, X'88D') Cluster name resolution failed.

MQRC_CLUSTER_RESOURCE_ERROR
 (2269, X'8DD') Cluster resource error.

MQRC_CONNECTION_BROKEN
 (2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
 (2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION QUIESCING
 (2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
 (2203, X'89B') Connection shutting down.

MQRC_DB2_NOT_AVAILABLE
 (2342, X'926') DB2 subsystem not available.

MQRC_DEF_XMIT_Q_TYPE_ERROR
 (2198, X'896') Default transmission queue not local.

MQRC_DEF_XMIT_Q_USAGE_ERROR
 (2199, X'897') Default transmission queue usage error.

MQRC_DYNAMIC_Q_NAME_ERROR
 (2011, X'7DB') Name of dynamic queue not valid.

MQRC_HANDLE_NOT_AVAILABLE
 (2017, X'7E1') No more handles available.

MQRC_HCONN_ERROR
 (2018, X'7E2') Connection handle not valid.

MQRC_HOBJ_ERROR
 (2019, X'7E3') Object handle not valid.

MQRC_MULTIPLE_REASONS
 (2136, X'858') Multiple reason codes returned.

MQRC_NAME_IN_USE
 (2201, X'899') Name in use.

MQOPEN – Reason parameter

MQRC_NAME_NOT_VALID_FOR_TYPE	(2194, X'892')	Object name not valid for object type.
MQRC_NOT_AUTHORIZED	(2035, X'7F3')	Not authorized for access.
MQRC_OBJECT_ALREADY_EXISTS	(2100, X'834')	Object already exists.
MQRC_OBJECT_DAMAGED	(2101, X'835')	Object damaged.
MQRC_OBJECT_IN_USE	(2042, X'7FA')	Object already open with conflicting options.
MQRC_OBJECT_LEVEL_INCOMPATIBLE	(2360, X'938')	Object level not compatible.
MQRC_OBJECT_NAME_ERROR	(2152, X'868')	Object name not valid.
MQRC_OBJECT_NOT_UNIQUE	(2343, X'927')	Object not unique.
MQRC_OBJECT_Q_MGR_NAME_ERROR	(2153, X'869')	Object queue-manager name not valid.
MQRC_OBJECT_RECORDS_ERROR	(2155, X'86B')	Object records not valid.
MQRC_OBJECT_TYPE_ERROR	(2043, X'7FB')	Object type not valid.
MQRC_OD_ERROR	(2044, X'7FC')	Object descriptor structure not valid.
MQRC_OPTION_NOT_VALID_FOR_TYPE	(2045, X'7FD')	Option not valid for object type.
MQRC_OPTIONS_ERROR	(2046, X'7FE')	Options not valid or not consistent.
MQRC_PAGESET_ERROR	(2193, X'891')	Error accessing page-set data set.
MQRC_PAGESET_FULL	(2192, X'890')	External storage medium is full.
MQRC_Q_DELETED	(2052, X'804')	Queue has been deleted.
MQRC_Q_MGR_NAME_ERROR	(2058, X'80A')	Queue manager name not valid or not known.
MQRC_Q_MGR_NOT_AVAILABLE	(2059, X'80B')	Queue manager not available for connection.
MQRC_Q_MGR QUIESCING	(2161, X'871')	Queue manager quiescing.
MQRC_Q_MGR STOPPING	(2162, X'872')	Queue manager shutting down.
MQRC_Q_TYPE_ERROR	(2057, X'809')	Queue type not valid.
MQRC_RECS_PRESENT_ERROR	(2154, X'86A')	Number of records present not valid.
MQRC_REMOTE_Q_NAME_ERROR	(2184, X'888')	Remote queue name not valid.
MQRC_RESOURCE_PROBLEM	(2102, X'836')	Insufficient system resources available.
MQRC_RESPONSE_RECORDS_ERROR	(2156, X'86C')	Response records not valid.
MQRC_SECURITY_ERROR	(2063, X'80F')	Security error occurred.
MQRC_STOPPED_BY_CLUSTER_EXIT	(2188, X'88C')	Call rejected by cluster workload exit.

MQRC_STORAGE_MEDIUM_FULL	(2192, X'890')	External storage medium is full.
MQRC_STORAGE_NOT_AVAILABLE	(2071, X'817')	Insufficient storage available.
MQRC_SUPPRESSED_BY_EXIT	(2109, X'83D')	Call suppressed by exit program.
MQRC_UNEXPECTED_ERROR	(2195, X'893')	Unexpected error occurred.
MQRC_UNKNOWN_ALIAS_BASE_Q	(2082, X'822')	Unknown alias base queue.
MQRC_UNKNOWN_DEF_XMIT_Q	(2197, X'895')	Unknown default transmission queue.
MQRC_UNKNOWN_OBJECT_NAME	(2085, X'825')	Unknown object name.
MQRC_UNKNOWN_OBJECT_Q_MGR	(2086, X'826')	Unknown object queue manager.
MQRC_UNKNOWN_REMOTE_Q_MGR	(2087, X'827')	Unknown remote queue manager.
MQRC_UNKNOWN_XMIT_Q	(2196, X'894')	Unknown transmission queue.
MQRC_WRONG_CF_LEVEL	(2366, X'93E')	Coupling-facility structure is wrong level.
MQRC_XMIT_Q_TYPE_ERROR	(2091, X'82B')	Transmission queue not local.
MQRC_XMIT_Q_USAGE_ERROR	(2092, X'82C')	Transmission queue with wrong usage.

For more information on these reason codes, see Appendix A, “Return codes”, on page 527.

Usage notes

- The object opened is one of the following:
 - A queue, in order to:
 - Get or browse messages (using the MQGET call)
 - Put messages (using the MQPUT call)
 - Inquire about the attributes of the queue (using the MQINQ call)
 - Set the attributes of the queue (using the MQSET call)

If the queue named is a model queue, a dynamic local queue is created. See the *ObjDesc* parameter described in Chapter 36, “MQOPEN – Open object”, on page 405.

A distribution list is a special type of queue object that contains a list of queues. It can be opened to put messages, but not to get or browse messages, or to inquire or set attributes. See usage note 8 for further details.

A queue that has QSGDISP(GROUP) is a special type of queue definition that cannot be used with the MQOPEN or MQPUT1 calls.

- A namelist, in order to:
 - Inquire about the names of the queues in the list (using the MQINQ call).
- A process definition, in order to:
 - Inquire about the process attributes (using the MQINQ call).
- The queue manager, in order to:

MQOPEN – Usage notes

- Inquire about the attributes of the local queue manager (using the MQINQ call).
- 2. It is valid for an application to open the same object more than once. A different object handle is returned for each open. Each handle that is returned can be used for the functions for which the corresponding open was performed.
- 3. If the object being opened is a queue but not a cluster queue, all name resolution within the local queue manager takes place at the time of the MQOPEN call. This may include one or more of the following for a given MQOPEN call:
 - Alias resolution to the name of a base queue
 - Resolution of the name of a local definition of a remote queue to the name of the remote queue manager, and the name by which the queue is known at the remote queue manager
 - Resolution of the remote queue-manager name to the name of a local transmission queue
 - (z/OS only) Resolution of the remote queue-manager name to the name of the shared transmission queue used by the IGQ agent (applies only if the local and remote queue managers belong to the same queue-sharing group)

However, be aware that subsequent MQINQ or MQSET calls for the handle relate solely to the name that has been opened, and not to the object resulting after name resolution has occurred. For example, if the object opened is an alias, the attributes returned by the MQINQ call are the attributes of the alias, not the attributes of the base queue to which the alias resolves. Name resolution checking is still carried out, however, regardless of what is specified for the *Options* parameter on the corresponding MQOPEN.

If the object being opened is a cluster queue, name resolution can occur at the time of the MQOPEN call, or be deferred until later. The point at which resolution occurs is controlled by the MQOO_BIND_* options specified on the MQOPEN call:

```
MQOO_BIND_ON_OPEN
MQOO_BIND_NOT_FIXED
MQOO_BIND_AS_Q_DEF
```

Refer to the *WebSphere MQ Queue Manager Clusters* book for more information about name resolution for cluster queues.

- 4. The attributes of an object can change while an application has the object open. In many cases, the application does not notice this, but for certain attributes the queue manager marks the handle as no longer valid. These are:
 - Any attribute that affects the name resolution of the object. This applies regardless of the open options used, and includes the following:
 - A change to the *BaseQName* attribute of an alias queue that is open.
 - A change to the *RemoteQName* or *RemoteQMGrName* queue attributes, for any handle that is open for this queue, or for a queue which resolves through this definition as a queue-manager alias.
 - Any change that causes a currently-open handle for a remote queue to resolve to a different *transmission* queue, or to fail to resolve to one at all. For example, this can include:
 - A change to the *XmitQName* attribute of the local definition of a remote queue, whether the definition is being used for a queue, or for a queue-manager alias.

- (z/OS only) A change to the value of the *IntraGroupQueuing* queue-manager attribute, or a change in the definition of the shared transmission queue (SYSTEM.QSG.TRANSMIT.QUEUE) used by the IGQ agent.

There is one exception to this, namely the creation of a new transmission queue. A handle that would have resolved to this queue had it been present when the handle was opened, but instead resolved to the default transmission queue, is not made invalid.

- A change to the *DefXmitQName* queue-manager attribute. In this case all open handles that resolved to the previously-named queue (that resolved to it only because it was the default transmission queue) are marked as invalid. Handles that resolved to this queue for other reasons are not affected.
- The *Shareability* queue attribute, if there are two or more handles that are currently providing MQOO_INPUT_SHARED access for this queue, or for a queue that resolves to this queue. If this is the case, *all* handles that are open for this queue, or for a queue that resolves to this queue, are marked as invalid, regardless of the open options.

On z/OS, the handles described above are marked as invalid if one or more handles is currently providing MQOO_INPUT_SHARED or MQOO_INPUT_EXCLUSIVE access to the queue.

- The *Usage* queue attribute, for all handles that are open for this queue, or for a queue that resolves to this queue, regardless of the open options.

When a handle is marked as invalid, all subsequent calls (other than MQCLOSE) using this handle fail with reason code MQRC_OBJECT_CHANGED; the application should issue an MQCLOSE call (using the original handle) and then reopen the queue. Any uncommitted updates against the old handle from previous successful calls can still be committed or backed out, as required by the application logic.

If changing an attribute will cause this to happen, a special “force” version of the command must be used.

5. The queue manager performs security checks when an MQOPEN call is issued, to verify that the user identifier under which the application is running has the appropriate level of authority before access is permitted. The authority check is made on the name of the object being opened, and not on the name, or names, resulting after a name has been resolved.

If the object being opened is a model queue, the queue manager performs a full security check against both the name of the model queue and the name of the dynamic queue that is created. If the resulting dynamic queue is subsequently opened explicitly, a further resource security check is performed against the name of the dynamic queue.

On z/OS, the queue manager performs security checks only if security is enabled. For more information on security checking, see the *WebSphere MQ for z/OS System Setup Guide*.

6. A remote queue can be specified in one of two ways in the *ObjDesc* parameter of this call (see the *ObjectName* and *ObjectQMgrName* fields described in Chapter 12, “MQOD – Object descriptor”, on page 211):
 - By specifying for *ObjectName* the name of a local definition of the remote queue. In this case, *ObjectQMgrName* refers to the local queue manager, and can be specified as blanks or (in the C programming language) a null string.

MQOPEN – Usage notes

The security validation performed by the local queue manager verifies that the user is authorized to open the local definition of the remote queue.

- By specifying for *ObjectName* the name of the remote queue as known to the remote queue manager. In this case, *ObjectQMgrName* is the name of the remote queue manager.

The security validation performed by the local queue manager verifies that the user is authorized to send messages to the transmission queue resulting from the name resolution process.

In either case:

- No messages are sent by the local queue manager to the remote queue manager in order to check that the user is authorized to put messages on the queue.
 - When a message arrives at the remote queue manager, the remote queue manager may reject it because the user originating the message is not authorized.
7. An MQOPEN call with the MQOO_BROWSE option establishes a browse cursor, for use with MQGET calls that specify the object handle and one of the browse options. This allows the queue to be scanned without altering its contents. A message that has been found by browsing can subsequently be removed from the queue by using the MQGMO_MSG_UNDER_CURSOR option.

Multiple browse cursors can be active for a single application by issuing several MQOPEN requests for the same queue.

8. The following notes apply to the use of distribution lists.

Distribution lists are supported in the following environments: AIX, HP-UX, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

- a. Fields in the MQOD structure must be set as follows when opening a distribution list:
- *Version* must be MQOD_VERSION_2 or greater.
 - *ObjectType* must be MQOT_Q.
 - *ObjectName* must be blank or the null string.
 - *ObjectQMgrName* must be blank or the null string.
 - *RecsPresent* must be greater than zero.
 - One of *ObjectRecOffset* and *ObjectRecPtr* must be zero and the other nonzero.
 - No more than one of *ResponseRecOffset* and *ResponseRecPtr* can be nonzero.
 - There must be *RecsPresent* object records, addressed by either *ObjectRecOffset* or *ObjectRecPtr*. The object records must be set to the names of the destination queues to be opened.
 - If one of *ResponseRecOffset* and *ResponseRecPtr* is nonzero, there must be *RecsPresent* response records present. They are set by the queue manager if the call completes with reason code MQRC_MULTIPLE_REASONS.

A version-2 MQOD can also be used to open a single queue that is not in a distribution list, by ensuring that *RecsPresent* is zero.

- b. Only the following open options are valid in the *Options* parameter:
- MQOO_OUTPUT
 - MQOO_PASS_*_CONTEXT
 - MQOO_SET_*_CONTEXT
 - MQOO_ALTERNATE_USER_AUTHORITY

MQOO_FAIL_IF_QUIESCING

- c. The destination queues in the distribution list can be local, alias, or remote queues, but they cannot be model queues. If a model queue is specified, that queue fails to open, with reason code MQRC_Q_TYPE_ERROR. However, this does not prevent other queues in the list being opened successfully.
- d. The completion code and reason code parameters are set as follows:
- If the open operations for the queues in the distribution list all succeed or fail in the same way, the completion code and reason code parameters are set to describe the common result. The MQRR response records (if provided by the application) are not set in this case. For example, if every open succeeds, the completion code and reason code are set to MQCC_OK and MQRC_NONE respectively; if every open fails because none of the queues exists, the parameters are set to MQCC_FAILED and MQRC_UNKNOWN_OBJECT_NAME.
 - If the open operations for the queues in the distribution list do not all succeed or fail in the same way:
 - The completion code parameter is set to MQCC_WARNING if at least one open succeeded, and to MQCC_FAILED if all failed.
 - The reason code parameter is set to MQRC_MULTIPLE_REASONS.
 - The response records (if provided by the application) are set to the individual completion codes and reason codes for the queues in the distribution list.
- e. When a distribution list has been opened successfully, the handle *Hobj* returned by the call can be used on subsequent MQPUT calls to put messages to queues in the distribution list, and on an MQCLOSE call to relinquish access to the distribution list. The only valid close option for a distribution list is MQCO_NONE. The MQPUT1 call can also be used to put a message to a distribution list; the MQOD structure defining the queues in the list is specified as a parameter on that call.
- f. Each successfully-opened destination in the distribution list counts as a *separate* handle when checking whether the application has exceeded the permitted maximum number of handles (see the *MaxHandles* queue-manager attribute). This is true even when two or more of the destinations in the distribution list actually resolve to the same physical queue. If the MQOPEN or MQPUT1 call for a distribution list would cause the number of handles in use by the application to exceed *MaxHandles*, the call fails with reason code MQRC_HANDLE_NOT_AVAILABLE. On Compaq NonStop Kernel, the *MaxHandles* attribute is ignored.
- g. Each destination that is opened successfully has the value of its *OpenOutputCount* attribute incremented by one. If two or more of the destinations in the distribution list actually resolve to the same physical queue, that queue has its *OpenOutputCount* attribute incremented by the number of destinations in the distribution list that resolve to that queue.
- h. Any change to the queue definitions that would have caused a handle to become invalid had the queues been opened individually (for example, a change in the resolution path), does not cause the distribution-list handle to become invalid. However, it does result in a failure for that particular queue when the distribution-list handle is used on a subsequent MQPUT call.
- i. It is valid for a distribution list to contain only one destination.
9. The following notes apply to the use of cluster queues.

MQOPEN – Usage notes

Cluster queues are supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems. (Cluster queues are not available in WebSphere Application Server embedded messaging using reduced function WebSphere MQ.)

a. When a cluster queue is opened for the first time, and the local queue manager is not a full repository queue manager, the local queue manager obtains information about the cluster queue from a full repository queue manager. When the network is busy, it may take several seconds for the local queue manager to receive the needed information from the repository queue manager. As a result, the application issuing the MQOPEN call may have to wait for up to 10 seconds before control returns from the MQOPEN call. If the local queue manager does not receive the needed information about the cluster queue within this time, the call fails with reason code MQRC_CLUSTER_RESOLUTION_ERROR.

b. When a cluster queue is opened and there are multiple instances of the queue in the cluster, the instance actually opened depends on the options specified on the MQOPEN call:

- If the options specified include any of the following:

MQOO_BROWSE
MQOO_INPUT_AS_Q_DEF
MQOO_INPUT_EXCLUSIVE
MQOO_INPUT_SHARED
MQOO_SET

the instance of the cluster queue opened is required to be the local instance. If there is no local instance of the queue, the MQOPEN call fails.

- If the options specified include none of the above, but do include one or both of the following:

MQOO_INQUIRE
MQOO_OUTPUT

the instance opened is the local instance if there is one, and a remote instance otherwise. The instance chosen by the queue manager can, however, be altered by a cluster workload exit (if there is one).

For more information about cluster queues, refer to the *WebSphere MQ Queue Manager Clusters* book.

10. Applications started by a trigger monitor are passed the name of the queue that is associated with the application when the application is started. This queue name can be specified in the *ObjDesc* parameter to open the queue. See the description of the MQTMC2 structure for further details.

11. On OS/400, applications running in compatibility mode are connected automatically to the queue manager by the first MQOPEN call issued by the application (if the application has not already connected to the queue manager by using the MQCONN call).

Applications not running in compatibility mode must issue the MQCONN or MQCONNX call to connect to the queue manager explicitly, before using the MQOPEN call to open an object.

12. On Compaq NonStop Kernel, for a FASTPATH application opening or closing a dynamic queue, MQSeries can start and end a TM/MP transaction in order to update audited databases. If the application has opened the TM/MP T-file (because it can initiate multiple transactions), ENDTRANSACTION is a no-waited operation, and the application receives a completion for the

transaction initiated by MQSeries. Review the design of your applications to determine if this is the case and verify that the logic handling completions can cope with ENDTRANSACTION completions that are caused by MQSeries.

Language invocations

The MQOPEN call is supported in the programming languages shown below.

C invocation

```
MQOPEN (Hconn, &ObjDesc, Options, &Hobj, &CompCode,
        &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;    /* Connection handle */
MQOD     ObjDesc;  /* Object descriptor */
MQLONG   Options;  /* Options that control the action of MQOPEN */
MQHOBJ   Hobj;     /* Object handle */
MQLONG   CompCode; /* Completion code */
MQLONG   Reason;   /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQOPEN' USING HCONN, OBJDESC, OPTIONS, HOBJ, COMPCODE,
                   REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN    PIC S9(9) BINARY.
** Object descriptor
01 OBJDESC.
   COPY CMQODV.
** Options that control the action of MQOPEN
01 OPTIONS  PIC S9(9) BINARY.
** Object handle
01 HOBJ     PIC S9(9) BINARY.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON   PIC S9(9) BINARY.
```

PL/I invocation

```
call MQOPEN (Hconn, ObjDesc, Options, Hobj, CompCode, Reason);
```

Declare the parameters as follows:

```
dc1 Hconn    fixed bin(31); /* Connection handle */
dc1 ObjDesc  like MQOD;    /* Object descriptor */
dc1 Options  fixed bin(31); /* Options that control the action of
                           MQOPEN */
dc1 Hobj     fixed bin(31); /* Object handle */
dc1 CompCode fixed bin(31); /* Completion code */
dc1 Reason   fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
CALL MQOPEN,(HCONN,OBJDESC,OPTIONS,HOBJ,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN     DS      F Connection handle
OBJDESC   CMQODA  , Object descriptor
OPTIONS   DS      F Options that control the action of MQOPEN
```

MQOPEN – Language invocations

HOBJ	DS	F	Object handle
COMPCODE	DS	F	Completion code
REASON	DS	F	Reason code qualifying COMPCODE

TAL invocation

```
INT(32) .EXT HConn;  
STRUCT .EXT ObjDesc(MQOD^Def);  
INT(32) Options; INT(32) .EXT Hobj;  
INT(32) .EXT CC;  
INT(32) .EXT Reason;  
  
CALL MQOPEN(HConn, ObjDesc, Options, Hobj, CC, Reason);
```

Visual Basic invocation

```
MQOPEN Hconn, ObjDesc, Options, Hobj, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn As Long 'Connection handle'  
Dim ObjDesc As MQOD 'Object descriptor'  
Dim Options As Long 'Options that control the action of MQOPEN'  
Dim Hobj As Long 'Object handle'  
Dim CompCode As Long 'Completion code'  
Dim Reason As Long 'Reason code qualifying CompCode'
```

Chapter 37. MQPUT – Put message

The MQPUT call puts a message on a queue or distribution list. The queue or distribution list must already be open.

Syntax

MQPUT (*Hconn*, *Hobj*, *MsgDesc*, *PutMsgOpts*, *BufferLength*,
Buffer, *CompCode*, *Reason*)

Parameters

The MQPUT call has the following parameters.

Hconn (MQHCONN) – input

Connection handle.

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on OS/400 for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Hobj (MQHOBJ) – input

Object handle.

This handle represents the queue to which the message is added. The value of *Hobj* was returned by a previous MQOPEN call that specified the MQOO_OUTPUT option.

MsgDesc (MQMD) – input/output

Message descriptor.

This structure describes the attributes of the message being sent, and receives information about the message after the put request is complete. See Chapter 10, “MQMD – Message descriptor”, on page 141 for details.

If the application provides a version-1 MQMD, the message data can be prefixed with an MQMDE structure in order to specify values for the fields that exist in the version-2 MQMD but not the version-1. The *Format* field in the MQMD must be set to MQFMT_MD_EXTENSION to indicate that an MQMDE is present. See Chapter 11, “MQMDE – Message descriptor extension”, on page 203 for more details.

MQPUT – PutMsgOpts parameter

PutMsgOpts (MQPMO) – input/output

Options that control the action of MQPUT.

See Chapter 14, “MQPMO – Put-message options”, on page 229 for details.

BufferLength (MQLONG) – input

Length of the message in *Buffer*.

Zero is valid, and indicates that the message contains no application data. The upper limit for *BufferLength* depends on various factors:

- If the destination queue is a shared queue, the upper limit is 63 KB (64 512 bytes).
- If the destination is a local queue or resolves to a local queue (but is not a shared queue), the upper limit depends on whether:
 - The local queue manager supports segmentation.
 - The sending application specifies the flag that allows the queue manager to segment the message. This flag is MQMF_SEGMENTATION_ALLOWED, and can be specified either in a version-2 MQMD, or in an MQMDE used with a version-1 MQMD.

If both of these conditions are satisfied, *BufferLength* cannot exceed 999 999 999 minus the value of the *Offset* field in MQMD. The longest logical message that can be put is therefore 999 999 999 bytes (when *Offset* is zero). However, resource constraints imposed by the operating system or environment in which the application is running may result in a lower limit.

If one or both of the above conditions is not satisfied, *BufferLength* cannot exceed the smaller of the queue’s *MaxMsgLength* attribute and queue-manager’s *MaxMsgLength* attribute.

- If the destination is a remote queue or resolves to a remote queue, the conditions for local queues apply, *but at each queue manager through which the message must pass in order to reach the destination queue*; in particular:
 1. The local transmission queue used to store the message temporarily at the local queue manager
 2. Intermediate transmission queues (if any) used to store the message at queue managers on the route between the local and destination queue managers
 3. The destination queue at the destination queue manager

The longest message that can be put is therefore governed by the most restrictive of these queues and queue managers.

When a message is on a transmission queue, additional information resides with the message data, and this reduces the amount of application data that can be carried. In this situation it is recommended that MQ_MSG_HEADER_LENGTH bytes be subtracted from the *MaxMsgLength* values of the transmission queues when determining the limit for *BufferLength*.

Note: Only failure to comply with condition 1 can be diagnosed synchronously (with reason code MQRC_MSG_TOO_BIG_FOR_Q or MQRC_MSG_TOO_BIG_FOR_Q_MGR) when the message is put. If conditions 2 or 3 are not satisfied, the message is redirected to a dead-letter (undelivered-message) queue, either at an intermediate queue manager or at the destination queue manager. If this happens, a report message is generated if one was requested by the sender.

Buffer (MQBYTE×BufferLength) – input

Message data.

This is a buffer containing the application data to be sent. The buffer should be aligned on a boundary appropriate to the nature of the data in the message. 4-byte alignment should be suitable for most messages (including messages containing MQ header structures), but some messages may require more stringent alignment. For example, a message containing a 64-bit binary integer might require 8-byte alignment.

If *Buffer* contains character and/or numeric data, the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter should be set to the values appropriate to the data; this will enable the receiver of the message to convert the data (if necessary) to the character set and encoding used by the receiver.

Note: All of the other parameters on the MQPUT call must be in the character set and encoding of the local queue manager (given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE, respectively).

In the C programming language, the parameter is declared as a pointer-to-void; this means that the address of any type of data can be specified as the parameter.

If the *BufferLength* parameter is zero, *Buffer* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler can be null.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_WARNING

Warning (partial completion).

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_INCOMPLETE_GROUP

(2241, X'8C1') Message group not complete.

MQRC_INCOMPLETE_MSG

(2242, X'8C2') Logical message not complete.

MQRC_INCONSISTENT_PERSISTENCE

(2185, X'889') Inconsistent persistence specification.

MQRC_INCONSISTENT_UOW

(2245, X'8C5') Inconsistent unit-of-work specification.

MQPUT – Reason parameter

MQRC_MULTIPLE_REASONS

(2136, X'858') Multiple reason codes returned.

MQRC_PRIORITY_EXCEEDS_MAXIMUM

(2049, X'801') Message Priority exceeds maximum value supported.

MQRC_UNKNOWN_REPORT_OPTION

(2104, X'838') Report option(s) in message descriptor not recognized.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE

(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR

(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR

(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR

(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH

(2157, X'86D') Primary and home ASIDs differ.

MQRC_BACKED_OUT

(2003, X'7D3') Unit of work backed out.

MQRC_BUFFER_ERROR

(2004, X'7D4') Buffer parameter not valid.

MQRC_BUFFER_LENGTH_ERROR

(2005, X'7D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS

(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_STRUC_FAILED

(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE

(2346, X'92A') Coupling-facility structure in use.

MQRC_CFH_ERROR

(2235, X'8BB') PCF header structure not valid.

MQRC_CFIL_ERROR

(2236, X'8BC') PCF integer list parameter structure not valid.

MQRC_CFIN_ERROR

(2237, X'8BD') PCF integer parameter structure not valid.

MQRC_CFSL_ERROR

(2238, X'8BE') PCF string list parameter structure not valid.

MQRC_CFST_ERROR

(2239, X'8BF') PCF string parameter structure not valid.

MQRC_CICS_WAIT_FAILED

(2140, X'85C') Wait request rejected by CICS.

MQRC_CLUSTER_EXIT_ERROR

(2266, X'8DA') Cluster workload exit failed.

MQRC_CLUSTER_RESOLUTION_ERROR

(2189, X'88D') Cluster name resolution failed.

MQRC_CLUSTER_RESOURCE_ERROR

(2269, X'8DD') Cluster resource error.

MQRC_COD_NOT_VALID_FOR_XCF_Q

(2106, X'83A') COD report option not valid for XCF queue.

MQRC_CONNECTION_BROKEN

(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED

(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION QUIESCING

(2202, X'89A') Connection quiescing.

	MQRC_CONNECTION_STOPPING	(2203, X'89B')	Connection shutting down.
	MQRC_CONTEXT_HANDLE_ERROR	(2097, X'831')	Queue handle referred to does not save context.
	MQRC_CONTEXT_NOT_AVAILABLE	(2098, X'832')	Context not available for queue handle referred to.
	MQRC_DATA_LENGTH_ERROR	(2010, X'7DA')	Data length parameter not valid.
	MQRC_DH_ERROR	(2135, X'857')	Distribution header structure not valid.
	MQRC_DLH_ERROR	(2141, X'85D')	Dead letter header structure not valid.
	MQRC_EXPIRY_ERROR	(2013, X'7DD')	Expiry time not valid.
	MQRC_FEEDBACK_ERROR	(2014, X'7DE')	Feedback code not valid.
	MQRC_GLOBAL_UOW_CONFLICT	(2351, X'92F')	Global units of work conflict.
	MQRC_GROUP_ID_ERROR	(2258, X'8D2')	Group identifier not valid.
	MQRC_HANDLE_IN_USE_FOR_UOW	(2353, X'931')	Handle in use for global unit of work.
	MQRC_HCONN_ERROR	(2018, X'7E2')	Connection handle not valid.
	MQRC_HEADER_ERROR	(2142, X'85E')	MQ header structure not valid.
	MQRC_HOBJ_ERROR	(2019, X'7E3')	Object handle not valid.
	MQRC_IH_ERROR	(2148, X'864')	IMS information header structure not valid.
	MQRC_INCOMPLETE_GROUP	(2241, X'8C1')	Message group not complete.
	MQRC_INCOMPLETE_MSG	(2242, X'8C2')	Logical message not complete.
	MQRC_INCONSISTENT_PERSISTENCE	(2185, X'889')	Inconsistent persistence specification.
	MQRC_INCONSISTENT_UOW	(2245, X'8C5')	Inconsistent unit-of-work specification.
	MQRC_LOCAL_UOW_CONFLICT	(2352, X'930')	Global unit of work conflicts with local unit of work.
	MQRC_MD_ERROR	(2026, X'7EA')	Message descriptor not valid.
	MQRC_MDE_ERROR	(2248, X'8C8')	Message descriptor extension not valid.
	MQRC_MISSING_REPLY_TO_Q	(2027, X'7EB')	Missing reply-to queue.
	MQRC_MISSING_WIH	(2332, X'91C')	Message data does not begin with MQWIH.
	MQRC_MSG_FLAGS_ERROR	(2249, X'8C9')	Message flags not valid.
	MQRC_MSG_SEQ_NUMBER_ERROR	(2250, X'8CA')	Message sequence number not valid.
	MQRC_MSG_TOO_BIG_FOR_Q	(2030, X'7EE')	Message length greater than maximum for queue.
	MQRC_MSG_TOO_BIG_FOR_Q_MGR	(2031, X'7EF')	Message length greater than maximum for queue manager.

MQPUT – Reason parameter

MQRC_MSG_TYPE_ERROR	(2029, X'7ED')	Message type in message descriptor not valid.
MQRC_MULTIPLE_REASONS	(2136, X'858')	Multiple reason codes returned.
MQRC_NO_DESTINATIONS_AVAILABLE	(2270, X'8DE')	No destination queues available.
MQRC_NOT_OPEN_FOR_OUTPUT	(2039, X'7F7')	Queue not open for output.
MQRC_NOT_OPEN_FOR_PASS_ALL	(2093, X'82D')	Queue not open for pass all context.
MQRC_NOT_OPEN_FOR_PASS_IDENT	(2094, X'82E')	Queue not open for pass identity context.
MQRC_NOT_OPEN_FOR_SET_ALL	(2095, X'82F')	Queue not open for set all context.
MQRC_NOT_OPEN_FOR_SET_IDENT	(2096, X'830')	Queue not open for set identity context.
MQRC_OBJECT_CHANGED	(2041, X'7F9')	Object definition changed since opened.
MQRC_OBJECT_DAMAGED	(2101, X'835')	Object damaged.
MQRC_OFFSET_ERROR	(2251, X'8CB')	Message segment offset not valid.
MQRC_OPEN_FAILED	(2137, X'859')	Object not opened successfully.
MQRC_OPTIONS_ERROR	(2046, X'7FE')	Options not valid or not consistent.
MQRC_ORIGINAL_LENGTH_ERROR	(2252, X'8CC')	Original length not valid.
MQRC_PAGESET_ERROR	(2193, X'891')	Error accessing page-set data set.
MQRC_PAGESET_FULL	(2192, X'890')	External storage medium is full.
MQRC_PCF_ERROR	(2149, X'865')	PCF structures not valid.
MQRC_PERSISTENCE_ERROR	(2047, X'7FF')	Persistence not valid.
MQRC_PERSISTENT_NOT_ALLOWED	(2048, X'800')	Queue does not support persistent messages.
MQRC_PMO_ERROR	(2173, X'87D')	Put-message options structure not valid.
MQRC_PMO_RECORD_FLAGS_ERROR	(2158, X'86E')	Put message record flags not valid.
MQRC_PRIORITY_ERROR	(2050, X'802')	Message priority not valid.
MQRC_PUT_INHIBITED	(2051, X'803')	Put calls inhibited for the queue.
MQRC_PUT_MSG_RECORDS_ERROR	(2159, X'86F')	Put message records not valid.
MQRC_Q_DELETED	(2052, X'804')	Queue has been deleted.
MQRC_Q_FULL	(2053, X'805')	Queue already contains maximum number of messages.
MQRC_Q_MGR_NAME_ERROR	(2058, X'80A')	Queue manager name not valid or not known.
MQRC_Q_MGR_NOT_AVAILABLE	(2059, X'80B')	Queue manager not available for connection.

MQRC_Q_MGR QUIESCING
 (2161, X'871') Queue manager quiescing.

MQRC_Q_MGR STOPPING
 (2162, X'872') Queue manager shutting down.

MQRC_Q_SPACE_NOT_AVAILABLE
 (2056, X'808') No space available on disk for queue.

MQRC_RECS_PRESENT_ERROR
 (2154, X'86A') Number of records present not valid.

MQRC_REPORT_OPTIONS_ERROR
 (2061, X'80D') Report options in message descriptor not valid.

MQRC_RESOURCE_PROBLEM
 (2102, X'836') Insufficient system resources available.

MQRC_RESPONSE_RECORDS_ERROR
 (2156, X'86C') Response records not valid.

MQRC_RFH_ERROR
 (2334, X'91E') MQRFH or MQRFH2 structure not valid.

MQRC_RMH_ERROR
 (2220, X'8AC') Reference message header structure not valid.

MQRC_SEGMENT_LENGTH_ZERO
 (2253, X'8CD') Length of data in message segment is zero.

MQRC_SEGMENTS_NOT_SUPPORTED
 (2365, X'93D') Segments not supported.

MQRC_STOPPED_BY_CLUSTER_EXIT
 (2188, X'88C') Call rejected by cluster workload exit.

MQRC_STORAGE_CLASS_ERROR
 (2105, X'839') Storage class error.

MQRC_STORAGE_MEDIUM_FULL
 (2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE
 (2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
 (2109, X'83D') Call suppressed by exit program.

MQRC_SYNCPOINT_LIMIT_REACHED
 (2024, X'7E8') No more messages can be handled within current unit of work.

MQRC_SYNCPOINT_NOT_AVAILABLE
 (2072, X'818') Syncpoint support not available.

MQRC_TM_ERROR
 (2265, X'8D9') Trigger message structure not valid.

MQRC_TMC_ERROR
 (2191, X'88F') Character trigger message structure not valid.

MQRC_UNEXPECTED_ERROR
 (2195, X'893') Unexpected error occurred.

MQRC_UOW_ENLISTMENT_ERROR
 (2354, X'932') Enlistment in global unit of work failed.

MQRC_UOW_MIX_NOT_SUPPORTED
 (2355, X'933') Mixture of unit-of-work calls not supported.

MQRC_UOW_NOT_AVAILABLE
 (2255, X'8CF') Unit of work not available for the queue manager to use.

MQRC_WIH_ERROR
 (2333, X'91D') MQWIH structure not valid.

MQRC_WRONG_MD_VERSION
 (2257, X'8D1') Wrong version of MQMD supplied.

MQRC_XQH_ERROR
 (2260, X'8D4') Transmission queue header structure not valid.

MQPUT – Reason parameter

For more information on these reason codes, see Appendix A, “Return codes”, on page 527.

Usage notes

1. Both the MQPUT and MQPUT1 calls can be used to put messages on a queue; which call to use depends on the circumstances:
 - The MQPUT call should be used when multiple messages are to be placed on the *same* queue.

An MQOPEN call specifying the MQOO_OUTPUT option is issued first, followed by one or more MQPUT requests to add messages to the queue; finally the queue is closed with an MQCLOSE call. This gives better performance than repeated use of the MQPUT1 call.
 - The MQPUT1 call should be used when only *one* message is to be put on a queue.

This call encapsulates the MQOPEN, MQPUT, and MQCLOSE calls into a single call, thereby minimizing the number of calls that must be issued.
2. If an application puts a sequence of messages on the same queue without using message groups, the order of those messages is preserved provided that the conditions detailed below are satisfied. Some conditions apply to both local and remote destination queues; other conditions apply only to remote destination queues.

Conditions that apply to local and remote destination queues

- All of the MQPUT calls are within the same unit of work, or none of them is within a unit of work.

Be aware that when messages are put onto a particular queue within a single unit of work, messages from other applications may be interspersed with the sequence of messages on the queue.
- All of the MQPUT calls are made using the same object handle *Hobj*.

In some environments, message sequence is also preserved when different object handles are used, provided the calls are made from the same application. The meaning of “same application” is determined by the environment:

 - On Compaq OpenVMS Alpha, the application is the thread.
 - On PC DOS, the application is the system.
 - On z/OS, the application is:
 - For CICS, the CICS task
 - For IMS, the task
 - For z/OS batch, the task
 - On OS/2, the application is the thread.
 - On OS/400, the application is the job.
 - On Compaq NonStop Kernel, the application is the thread.
 - On UNIX systems, the application is the thread.
 - On VSE/ESA, the application is the CICS task.
 - On Windows 3.1, the application is the process.
 - On Windows and Windows 95, Windows 98, the application is the thread.
- The messages all have the same priority.

Additional conditions that apply to remote destination queues

- There is only one path from the sending queue manager to the destination queue manager.

If there is a possibility that some messages in the sequence may go on a different path (for example, because of reconfiguration, traffic balancing, or

path selection based on message size), the order of the messages at the destination queue manager cannot be guaranteed.

- Messages are not placed temporarily on dead-letter queues at the sending, intermediate, or destination queue managers.

If one or more of the messages is put temporarily on a dead-letter queue (for example, because a transmission queue or the destination queue is temporarily full), the messages can arrive on the destination queue out of sequence.

- The messages are either all persistent or all nonpersistent.

If a channel on the route between the sending and destination queue managers has its *NonPersistentMsgSpeed* attribute set to MQNPMS_FAST, nonpersistent messages can jump ahead of persistent messages, resulting in the order of persistent messages relative to nonpersistent messages not being preserved. However, the order of persistent messages relative to each other, and of nonpersistent messages relative to each other, is preserved.

If these conditions are not satisfied, message groups can be used to preserve message order, but note that this requires both the sending and receiving applications to use the message-grouping support. For more information about message groups, see:

- *MsgFlags* field in MQMD
- MQPMO_LOGICAL_ORDER option in MQPMO
- MQGMO_LOGICAL_ORDER option in MQGMO

3. The following notes apply to the use of distribution lists.

Distribution lists are supported in the following environments: AIX, HP-UX, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

- a. Messages can be put to a distribution list using either a version-1 or a version-2 MQPMO. If a version-1 MQPMO is used (or a version-2 MQPMO with *RecsPresent* equal to zero), no put message records or response records can be provided by the application. This means that it will not be possible to identify the queues which encounter errors, if the message is sent successfully to some queues in the distribution list and not others.

If put message records or response records are provided by the application, the *Version* field must be set to MQPMO_VERSION_2.

A version-2 MQPMO can also be used to send messages to a single queue that is not in a distribution list, by ensuring that *RecsPresent* is zero.

- b. The completion code and reason code parameters are set as follows:

- If the puts to the queues in the distribution list all succeed or fail in the same way, the completion code and reason code parameters are set to describe the common result. The MQRR response records (if provided by the application) are not set in this case.

For example, if every put succeeds, the completion code and reason code are set to MQCC_OK and MQRC_NONE respectively; if every put fails because all of the queues are inhibited for puts, the parameters are set to MQCC_FAILED and MQRC_PUT_INHIBITED.

- If the puts to the queues in the distribution list do not all succeed or fail in the same way:
 - The completion code parameter is set to MQCC_WARNING if at least one put succeeded, and to MQCC_FAILED if all failed.
 - The reason code parameter is set to MQRC_MULTIPLE_REASONS.

MQPUT – Usage notes

- The response records (if provided by the application) are set to the individual completion codes and reason codes for the queues in the distribution list.

If the put to a destination fails because the open for that destination failed, the fields in the response record are set to MQCC_FAILED and MQRC_OPEN_FAILED; that destination is included in *InvalidDestCount*.

- c. If a destination in the distribution list resolves to a local queue, the message is placed on that queue in normal form (that is, not as a distribution-list message). If more than one destination resolves to the same local queue, one message is placed on the queue for each such destination.

If a destination in the distribution list resolves to a remote queue, a message is placed on the appropriate transmission queue. Where several destinations resolve to the same transmission queue, a single distribution-list message containing those destinations may be placed on the transmission queue, even if those destinations were not adjacent in the list of destinations provided by the application. However, this can be done only if the transmission queue supports distribution-list messages (see the *DistLists* queue attribute described in Chapter 40, “Attributes for queues”, on page 457).

If the transmission queue does not support distribution lists, one copy of the message in normal form is placed on the transmission queue for each destination that uses that transmission queue.

If a distribution list with the application message data is too big for a transmission queue, the distribution list message is split up into smaller distribution-list messages, each containing fewer destinations. If the application message data only just fits on the queue, distribution-list messages cannot be used at all, and the queue manager generates one copy of the message in normal form for each destination that uses that transmission queue.

If different destinations have different message priority or message persistence (this can occur when the application specifies MQPRI_PRIORITY_AS_Q_DEF or MQPER_PERSISTENCE_AS_Q_DEF), the messages are not held in the same distribution-list message. Instead, the queue manager generates as many distribution-list messages as are necessary to accommodate the differing priority and persistence values.

- d. A put to a distribution list may result in:
 - A single distribution-list message, or
 - A number of smaller distribution-list messages, or
 - A mixture of distribution list messages and normal messages, or
 - Normal messages only.

Which of the above occurs depends on whether:

- The destinations in the list are local, remote, or a mixture.
- The destinations have the same message priority and message persistence.
- The transmission queues can hold distribution-list messages.
- The transmission queues’ maximum message lengths are large enough to accommodate the message in distribution-list form.

However, regardless of which of the above occurs, each *physical* message resulting (that is, each normal message or distribution-list message resulting from the put) counts as only *one* message when:

- Checking whether the application has exceeded the permitted maximum number of messages in a unit of work (see the *MaxUncommittedMsgs* queue-manager attribute).
 - Checking whether the triggering conditions are satisfied.
 - Incrementing queue depths and checking whether the queues' maximum queue depth would be exceeded.
- e. Any change to the queue definitions that would have caused a handle to become invalid had the queues been opened individually (for example, a change in the resolution path), does not cause the distribution-list handle to become invalid. However, it does result in a failure for that particular queue when the distribution-list handle is used on a subsequent MQPUT call.
4. If a message is put with one or more MQ header structures at the beginning of the application message data, the queue manager performs certain checks on the header structures to verify that they are valid. If the queue manager detects an error, the call fails with an appropriate reason code. The checks performed vary according to the particular structures that are present:
- Checks are performed only if a version-2 or later MQMD is used on the MQPUT or MQPUT1 call. Checks are not performed if a version-1 MQMD is used, even if an MQMDE is present at the start of the message data.
 - Structures that are not supported by the local queue manager, and structures following the first MQDLH in the message, are not validated.
 - The MQDH and MQMDE structures are validated completely by the queue manager.
 - Other structures are validated partially by the queue manager (not all fields are checked).

General checks performed by the queue manager include the following:

- The *StrucId* field must be valid.
- The *Version* field must be valid.
- The *StrucLength* field must specify a value that is large enough to include the structure plus any variable-length data that forms part of the structure.
- The *CodedCharSetId* field must not be zero, or a negative value that is not valid (note: MQCCSI_DEFAULT, MQCCSI_EMBEDDED, MQCCSI_Q_MGR, and MQCCSI_UNDEFINED are *not* valid in most MQ header structures).
- The *BufferLength* parameter of the call must specify a value that is large enough to include the structure (the structure must not extend beyond the end of the message).

In addition to general checks on structures, the following conditions must be satisfied:

- The sum of the lengths of the structures in a PCF message must equal the length specified by the *BufferLength* parameter on the MQPUT or MQPUT1 call. A PCF message is a message that has a format name of MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF.
- An MQ structure must not be truncated, except in the following situations where truncated structures are permitted:
 - Messages which are report messages.
 - PCF messages.
 - Messages containing an MQDLH structure. (Structures *following* the first MQDLH can be truncated; structures preceding the MQDLH cannot.)
- An MQ structure must not be split over two or more segments – the structure must be contained entirely within one segment.

MQPUT – Usage notes

- For the Visual Basic programming language, the following points apply:
 - If the size of the *Buffer* parameter is less than the length specified by the *BufferLength* parameter, the call fails with reason code MQRC_BUFFER_LENGTH_ERROR.
 - The *Buffer* parameter is declared as being of type String. If the data to be placed on the queue is not of type String, the MQPUTAny call should be used in place of MQPUT.

The MQPUTAny call has the same parameters as the MQPUT call, except that the *Buffer* parameter is declared as being of type Any, allowing any type of data to be placed on the queue. However, this means that *Buffer* cannot be checked to ensure that it is at least *BufferLength* bytes in size.

- On Compaq NonStop Kernel, if the MQPUT call is issued outside a Compaq TMF transaction *without* the MQPMO_NO_SYNCPOINT option, the reason code MQRC_UNIT_OF_WORK_NOT_STARTED is returned.

Language invocations

The MQPUT call is supported in the programming languages shown below.

C invocation

```
MQPUT (Hconn, Hobj, &MsgDesc, &PutMsgOpts, BufferLength, Buffer,  
       &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;          /* Connection handle */  
MQHOBJ   Hobj;          /* Object handle */  
MQMD     MsgDesc;      /* Message descriptor */  
MQPMO    PutMsgOpts;   /* Options that control the action of MQPUT */  
MQLONG   BufferLength; /* Length of the message in Buffer */  
MQBYTE   Buffer[n];    /* Message data */  
MQLONG   CompCode;    /* Completion code */  
MQLONG   Reason;      /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQPUT' USING HCONN, HOBJ, MSGDESC, PUTMSGOPTS, BUFFERLENGTH,  
                  BUFFER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle  
01 HCONN          PIC S9(9) BINARY.  
** Object handle  
01 HOBJ          PIC S9(9) BINARY.  
** Message descriptor  
01 MSGDESC.  
   COPY CMQMDV.  
** Options that control the action of MQPUT  
01 PUTMSGOPTS.  
   COPY CMQPMOV.  
** Length of the message in BUFFER  
01 BUFFERLENGTH PIC S9(9) BINARY.  
** Message data  
01 BUFFER       PIC X(n).  
** Completion code  
01 COMPCODE     PIC S9(9) BINARY.  
** Reason code qualifying COMPCODE  
01 REASON      PIC S9(9) BINARY.
```

PL/I invocation

```
call MQPUT (Hconn, Hobj, MsgDesc, PutMsgOpts, BufferLength, Buffer,
           CompCode, Reason);
```

Declare the parameters as follows:

```
dc1 Hconn      fixed bin(31); /* Connection handle */
dc1 Hobj       fixed bin(31); /* Object handle */
dc1 MsgDesc    like MQMD;    /* Message descriptor */
dc1 PutMsgOpts like MQPMO;    /* Options that control the action of
                             MQPUT */
dc1 BufferLength fixed bin(31); /* Length of the message in Buffer */
dc1 Buffer      char(n);      /* Message data */
dc1 CompCode   fixed bin(31); /* Completion code */
dc1 Reason     fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
CALL MQPUT, (HCONN,HOBJ,MSGDESC,PUTMSGOPTS,BUFFERLENGTH,      X
            BUFFER,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN      DS      F      Connection handle
HOBJ       DS      F      Object handle
MSGDESC    CMQMDA   ,      Message descriptor
PUTMSGOPTS CMQPMOA  ,      Options that control the action of MQPUT
BUFFERLENGTH DS     F      Length of the message in BUFFER
BUFFER     DS      CL(n)  Message data
COMPCODE   DS      F      Completion code
REASON     DS      F      Reason code qualifying COMPCODE
```

TAL invocation

```
INT(32) .EXT HConn;
INT(32) .EXT Hobj;
STRUCT  .EXT MsgDesc(MQMD^Def);
STRUCT  .EXT PutMsgOpt(MQPMO^Def);
INT(32) .EXT BufferLen
STRING  .EXT Buffer[0:BUFFER^SIZE]
INT(32) .EXT CC;
INT(32) .EXT Reason;
```

```
CALL MQPUT(HConn, Hobj, MsgDesc, PutMsgOpt, BufferLen, Buffer,
           CC, Reason);
```

Visual Basic invocation

```
MQPUT Hconn, Hobj, MsgDesc, PutMsgOpts, BufferLength, Buffer, CompCode,
      Reason
```

Declare the parameters as follows:

```
Dim Hconn      As Long 'Connection handle'
Dim Hobj       As Long 'Object handle'
Dim MsgDesc    As MQMD 'Message descriptor'
Dim PutMsgOpts As MQPMO 'Options that control the action of MQPUT'
Dim BufferLength As Long 'Length of the message in Buffer'
Dim Buffer      As String 'Message data'
Dim CompCode   As Long 'Completion code'
Dim Reason     As Long 'Reason code qualifying CompCode'
```

MQPUT – Language invocations

Chapter 38. MQPUT1 – Put one message

The MQPUT1 call puts one message on a queue. The queue need not be open.

Syntax

MQPUT1 (*Hconn*, *ObjDesc*, *MsgDesc*, *PutMsgOpts*, *BufferLength*,
Buffer, *CompCode*, *Reason*)

Parameters

The MQPUT1 call has the following parameters.

Hconn (MQHCONN) – input

Connection handle.

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on OS/400 for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

ObjDesc (MQOD) – input/output

Object descriptor.

This is a structure which identifies the queue to which the message is added. See Chapter 12, “MQOD – Object descriptor”, on page 211 for details.

The user must be authorized to open the queue for output. The queue must **not** be a model queue.

MsgDesc (MQMD) – input/output

Message descriptor.

This structure describes the attributes of the message being sent, and receives feedback information after the put request is complete. See Chapter 10, “MQMD – Message descriptor”, on page 141 for details.

If the application provides a version-1 MQMD, the message data can be prefixed with an MQMDE structure in order to specify values for the fields that exist in the version-2 MQMD but not the version-1. The *Format* field in the MQMD must be set to MQFMT_MD_EXTENSION to indicate that an MQMDE is present. See Chapter 11, “MQMDE – Message descriptor extension”, on page 203 for more details.

MQPUT1 – PutMsgOpts parameter

PutMsgOpts (MQPMO) – input/output

Options that control the action of MQPUT1.

See Chapter 14, “MQPMO – Put-message options”, on page 229 for details.

BufferLength (MQLONG) – input

Length of the message in *Buffer*.

Zero is valid, and indicates that the message contains no application data. The upper limit depends on various factors; see the description of the *BufferLength* parameter of the MQPUT call for further details.

Buffer (MQBYTE×BufferLength) – input

Message data.

This is a buffer containing the application message data to be sent. The buffer should be aligned on a boundary appropriate to the nature of the data in the message. 4-byte alignment should be suitable for most messages (including messages containing MQ header structures), but some messages may require more stringent alignment. For example, a message containing a 64-bit binary integer might require 8-byte alignment.

If *Buffer* contains character and/or numeric data, the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter should be set to the values appropriate to the data; this will enable the receiver of the message to convert the data (if necessary) to the character set and encoding used by the receiver.

Note: All of the other parameters on the MQPUT1 call must be in the character set and encoding of the local queue manager (given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE, respectively).

In the C programming language, the parameter is declared as a pointer-to-void; this means that the address of any type of data can be specified as the parameter.

If the *BufferLength* parameter is zero, *Buffer* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler can be null.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_WARNING

Warning (partial completion).

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:**MQRC_MULTIPLE_REASONS**

(2136, X'858') Multiple reason codes returned.

MQRC_INCOMPLETE_GROUP

(2241, X'8C1') Message group not complete.

MQRC_INCOMPLETE_MSG

(2242, X'8C2') Logical message not complete.

MQRC_PRIORITY_EXCEEDS_MAXIMUM

(2049, X'801') Message Priority exceeds maximum value supported.

MQRC_UNKNOWN_REPORT_OPTION

(2104, X'838') Report option(s) in message descriptor not recognized.

If *CompCode* is MQCC_FAILED:**MQRC_ADAPTER_NOT_AVAILABLE**

(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR

(2130, X'852') Unable to load adapter service module.

MQRC_ALIAS_BASE_Q_TYPE_ERROR

(2001, X'7D1') Alias base queue not a valid type.

MQRC_API_EXIT_ERROR

(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR

(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH

(2157, X'86D') Primary and home ASIDs differ.

MQRC_BACKED_OUT

(2003, X'7D3') Unit of work backed out.

MQRC_BUFFER_ERROR

(2004, X'7D4') Buffer parameter not valid.

MQRC_BUFFER_LENGTH_ERROR

(2005, X'7D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS

(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_NOT_AVAILABLE

(2345, X'929') Coupling facility not available.

MQRC_CF_STRUC_AUTH_FAILED

(2348, X'92C') Coupling-facility structure authorization check failed.

MQRC_CF_STRUC_ERROR

(2349, X'92D') Coupling-facility structure not valid.

MQRC_CF_STRUC_FAILED

(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE

(2346, X'92A') Coupling-facility structure in use.

MQRC_CF_STRUC_LIST_HDR_IN_USE

(2347, X'92B') Coupling-facility structure list-header in use.

MQRC_CFH_ERROR

(2235, X'8BB') PCF header structure not valid.

MQRC_CFIL_ERROR

(2236, X'8BC') PCF integer list parameter structure not valid.

MQRC_CFIN_ERROR

(2237, X'8BD') PCF integer parameter structure not valid.

MQRC_CFSL_ERROR

(2238, X'8BE') PCF string list parameter structure not valid.

MQPUT1 – Reason parameter

	MQRC_CFST_ERROR
	(2239, X'8BF') PCF string parameter structure not valid.
	MQRC_CICS_WAIT_FAILED
	(2140, X'85C') Wait request rejected by CICS.
	MQRC_CLUSTER_EXIT_ERROR
	(2266, X'8DA') Cluster workload exit failed.
	MQRC_CLUSTER_RESOLUTION_ERROR
	(2189, X'88D') Cluster name resolution failed.
	MQRC_CLUSTER_RESOURCE_ERROR
	(2269, X'8DD') Cluster resource error.
	MQRC_COD_NOT_VALID_FOR_XCF_Q
	(2106, X'83A') COD report option not valid for XCF queue.
	MQRC_CONNECTION_BROKEN
	(2009, X'7D9') Connection to queue manager lost.
	MQRC_CONNECTION_NOT_AUTHORIZED
	(2217, X'8A9') Not authorized for connection.
	MQRC_CONNECTION QUIESCING
	(2202, X'89A') Connection quiescing.
	MQRC_CONNECTION_STOPPING
	(2203, X'89B') Connection shutting down.
	MQRC_CONTEXT_HANDLE_ERROR
	(2097, X'831') Queue handle referred to does not save context.
	MQRC_CONTEXT_NOT_AVAILABLE
	(2098, X'832') Context not available for queue handle referred to.
	MQRC_DATA_LENGTH_ERROR
	(2010, X'7DA') Data length parameter not valid.
	MQRC_DB2_NOT_AVAILABLE
	(2342, X'926') DB2 subsystem not available.
	MQRC_DEF_XMIT_Q_TYPE_ERROR
	(2198, X'896') Default transmission queue not local.
	MQRC_DEF_XMIT_Q_USAGE_ERROR
	(2199, X'897') Default transmission queue usage error.
	MQRC_DH_ERROR
	(2135, X'857') Distribution header structure not valid.
	MQRC_DLH_ERROR
	(2141, X'85D') Dead letter header structure not valid.
	MQRC_EXPIRY_ERROR
	(2013, X'7DD') Expiry time not valid.
	MQRC_FEEDBACK_ERROR
	(2014, X'7DE') Feedback code not valid.
	MQRC_GLOBAL_UOW_CONFLICT
	(2351, X'92F') Global units of work conflict.
	MQRC_GROUP_ID_ERROR
	(2258, X'8D2') Group identifier not valid.
	MQRC_HANDLE_IN_USE_FOR_UOW
	(2353, X'931') Handle in use for global unit of work.
	MQRC_HANDLE_NOT_AVAILABLE
	(2017, X'7E1') No more handles available.
	MQRC_HCONN_ERROR
	(2018, X'7E2') Connection handle not valid.
	MQRC_HEADER_ERROR
	(2142, X'85E') MQ header structure not valid.
	MQRC_IH_ERROR
	(2148, X'864') IMS information header structure not valid.
	MQRC_LOCAL_UOW_CONFLICT
	(2352, X'930') Global unit of work conflicts with local unit of work.

MQRC_MD_ERROR	(2026, X'7EA') Message descriptor not valid.
MQRC_MDE_ERROR	(2248, X'8C8') Message descriptor extension not valid.
MQRC_MISSING_REPLY_TO_Q	(2027, X'7EB') Missing reply-to queue.
MQRC_MISSING_WIH	(2332, X'91C') Message data does not begin with MQWIH.
MQRC_MSG_FLAGS_ERROR	(2249, X'8C9') Message flags not valid.
MQRC_MSG_SEQ_NUMBER_ERROR	(2250, X'8CA') Message sequence number not valid.
MQRC_MSG_TOO_BIG_FOR_Q	(2030, X'7EE') Message length greater than maximum for queue.
MQRC_MSG_TOO_BIG_FOR_Q_MGR	(2031, X'7EF') Message length greater than maximum for queue manager.
MQRC_MSG_TYPE_ERROR	(2029, X'7ED') Message type in message descriptor not valid.
MQRC_MULTIPLE_REASONS	(2136, X'858') Multiple reason codes returned.
MQRC_NO_DESTINATIONS_AVAILABLE	(2270, X'8DE') No destination queues available.
MQRC_NOT_AUTHORIZED	(2035, X'7F3') Not authorized for access.
MQRC_OBJECT_DAMAGED	(2101, X'835') Object damaged.
MQRC_OBJECT_IN_USE	(2042, X'7FA') Object already open with conflicting options.
MQRC_OBJECT_LEVEL_INCOMPATIBLE	(2360, X'938') Object level not compatible.
MQRC_OBJECT_NAME_ERROR	(2152, X'868') Object name not valid.
MQRC_OBJECT_NOT_UNIQUE	(2343, X'927') Object not unique.
MQRC_OBJECT_Q_MGR_NAME_ERROR	(2153, X'869') Object queue-manager name not valid.
MQRC_OBJECT_RECORDS_ERROR	(2155, X'86B') Object records not valid.
MQRC_OBJECT_TYPE_ERROR	(2043, X'7FB') Object type not valid.
MQRC_OD_ERROR	(2044, X'7FC') Object descriptor structure not valid.
MQRC_OFFSET_ERROR	(2251, X'8CB') Message segment offset not valid.
MQRC_OPTIONS_ERROR	(2046, X'7FE') Options not valid or not consistent.
MQRC_ORIGINAL_LENGTH_ERROR	(2252, X'8CC') Original length not valid.
MQRC_PAGESET_ERROR	(2193, X'891') Error accessing page-set data set.
MQRC_PAGESET_FULL	(2192, X'890') External storage medium is full.
MQRC_PCF_ERROR	(2149, X'865') PCF structures not valid.
MQRC_PERSISTENCE_ERROR	(2047, X'7FF') Persistence not valid.

MQPUT1 – Reason parameter

MQRC_PERSISTENT_NOT_ALLOWED	(2048, X'800') Queue does not support persistent messages.
MQRC_PMO_ERROR	(2173, X'87D') Put-message options structure not valid.
MQRC_PMO_RECORD_FLAGS_ERROR	(2158, X'86E') Put message record flags not valid.
MQRC_PRIORITY_ERROR	(2050, X'802') Message priority not valid.
MQRC_PUT_INHIBITED	(2051, X'803') Put calls inhibited for the queue.
MQRC_PUT_MSG_RECORDS_ERROR	(2159, X'86F') Put message records not valid.
MQRC_Q_DELETED	(2052, X'804') Queue has been deleted.
MQRC_Q_FULL	(2053, X'805') Queue already contains maximum number of messages.
MQRC_Q_MGR_NAME_ERROR	(2058, X'80A') Queue manager name not valid or not known.
MQRC_Q_MGR_NOT_AVAILABLE	(2059, X'80B') Queue manager not available for connection.
MQRC_Q_MGR QUIESCING	(2161, X'871') Queue manager quiescing.
MQRC_Q_MGR_STOPPING	(2162, X'872') Queue manager shutting down.
MQRC_Q_SPACE_NOT_AVAILABLE	(2056, X'808') No space available on disk for queue.
MQRC_Q_TYPE_ERROR	(2057, X'809') Queue type not valid.
MQRC_RECS_PRESENT_ERROR	(2154, X'86A') Number of records present not valid.
MQRC_REMOTE_Q_NAME_ERROR	(2184, X'888') Remote queue name not valid.
MQRC_REPORT_OPTIONS_ERROR	(2061, X'80D') Report options in message descriptor not valid.
MQRC_RESOURCE_PROBLEM	(2102, X'836') Insufficient system resources available.
MQRC_RESPONSE_RECORDS_ERROR	(2156, X'86C') Response records not valid.
MQRC_RFH_ERROR	(2334, X'91E') MQRFH or MQRFH2 structure not valid.
MQRC_RMH_ERROR	(2220, X'8AC') Reference message header structure not valid.
MQRC_SECURITY_ERROR	(2063, X'80F') Security error occurred.
MQRC_SEGMENT_LENGTH_ZERO	(2253, X'8CD') Length of data in message segment is zero.
MQRC_STOPPED_BY_CLUSTER_EXIT	(2188, X'88C') Call rejected by cluster workload exit.
MQRC_STORAGE_CLASS_ERROR	(2105, X'839') Storage class error.
MQRC_STORAGE_MEDIUM_FULL	(2192, X'890') External storage medium is full.
MQRC_STORAGE_NOT_AVAILABLE	(2071, X'817') Insufficient storage available.
MQRC_SUPPRESSED_BY_EXIT	(2109, X'83D') Call suppressed by exit program.

	MQRC_SYNCPOINT_LIMIT_REACHED (2024, X'7E8')	No more messages can be handled within current unit of work.
	MQRC_SYNCPOINT_NOT_AVAILABLE (2072, X'818')	Syncpoint support not available.
	MQRC_TM_ERROR (2265, X'8D9')	Trigger message structure not valid.
	MQRC_TMC_ERROR (2191, X'88F')	Character trigger message structure not valid.
	MQRC_UNEXPECTED_ERROR (2195, X'893')	Unexpected error occurred.
	MQRC_UNKNOWN_ALIAS_BASE_Q (2082, X'822')	Unknown alias base queue.
	MQRC_UNKNOWN_DEF_XMIT_Q (2197, X'895')	Unknown default transmission queue.
	MQRC_UNKNOWN_OBJECT_NAME (2085, X'825')	Unknown object name.
	MQRC_UNKNOWN_OBJECT_Q_MGR (2086, X'826')	Unknown object queue manager.
	MQRC_UNKNOWN_REMOTE_Q_MGR (2087, X'827')	Unknown remote queue manager.
	MQRC_UNKNOWN_XMIT_Q (2196, X'894')	Unknown transmission queue.
	MQRC_UOW_ENLISTMENT_ERROR (2354, X'932')	Enlistment in global unit of work failed.
	MQRC_UOW_MIX_NOT_SUPPORTED (2355, X'933')	Mixture of unit-of-work calls not supported.
	MQRC_UOW_NOT_AVAILABLE (2255, X'8CF')	Unit of work not available for the queue manager to use.
	MQRC_WIH_ERROR (2333, X'91D')	MQWIH structure not valid.
	MQRC_WRONG_CF_LEVEL (2366, X'93E')	Coupling-facility structure is wrong level.
	MQRC_WRONG_MD_VERSION (2257, X'8D1')	Wrong version of MQMD supplied.
	MQRC_XMIT_Q_TYPE_ERROR (2091, X'82B')	Transmission queue not local.
	MQRC_XMIT_Q_USAGE_ERROR (2092, X'82C')	Transmission queue with wrong usage.
	MQRC_XQH_ERROR (2260, X'8D4')	Transmission queue header structure not valid.

For more information on these reason codes, see Appendix A, “Return codes”, on page 527.

Usage notes

- Both the MQPUT and MQPUT1 calls can be used to put messages on a queue; which call to use depends on the circumstances:
 - The MQPUT call should be used when multiple messages are to be placed on the *same* queue.
An MQOPEN call specifying the MQOO_OUTPUT option is issued first, followed by one or more MQPUT requests to add messages to the queue; finally the queue is closed with an MQCLOSE call. This gives better performance than repeated use of the MQPUT1 call.

MQPUT1 – Usage notes

- The MQPUT1 call should be used when only *one* message is to be put on a queue.

This call encapsulates the MQOPEN, MQPUT, and MQCLOSE calls into a single call, thereby minimizing the number of calls that must be issued.

2. If an application puts a sequence of messages on the same queue without using message groups, the order of those messages is preserved provided that certain conditions are satisfied. However, in most environments the MQPUT1 call does not satisfy these conditions, and so does not preserve message order. The MQPUT call must be used instead in these environments. See the usage notes in the description of the MQPUT call for details.
3. The MQPUT1 call can be used to put messages to distribution lists. For general information about this, see usage note 8 on page 418 for the MQOPEN call, and usage note 3 on page 431 for the MQPUT call.

Distribution lists are supported in the following environments: AIX, HP-UX, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

The following differences apply when using the MQPUT1 call:

- a. If MQRR response records are provided by the application, they must be provided using the MQOD structure; they cannot be provided using the MQPMO structure.
- b. The reason code MQRC_OPEN_FAILED is never returned by MQPUT1 in the response records; if a queue fails to open, the response record for that queue contains the actual reason code resulting from the open operation.

If an open operation for a queue succeeds with a completion code of MQCC_WARNING, the completion code and reason code in the response record for that queue are replaced by the completion and reason codes resulting from the put operation.

As with the MQOPEN and MQPUT calls, the queue manager sets the response records (if provided) only when the outcome of the call is not the same for all queues in the distribution list; this is indicated by the call completing with reason code MQRC_MULTIPLE_REASONS.

4. If the MQPUT1 call is used to put a message on a cluster queue, the call behaves as though MQOO_BIND_NOT_FIXED had been specified on the MQOPEN call.
5. If a message is put with one or more MQ header structures at the beginning of the application message data, the queue manager performs certain checks on the header structures to verify that they are valid. For more information about this, see usage note 4 on page 433 for the MQPUT call.
6. If more than one of the warning situations arise (see the *CompCode* parameter), the reason code returned is the *first* one in the following list that applies:
 - a. MQRC_MULTIPLE_REASONS
 - b. MQRC_INCOMPLETE_MSG
 - c. MQRC_INCOMPLETE_GROUP
 - d. MQRC_PRIORITY_EXCEEDS_MAXIMUM or MQRC_UNKNOWN_REPORT_OPTION
7. For the Visual Basic programming language, the following points apply:
 - If the size of the *Buffer* parameter is less than the length specified by the *BufferLength* parameter, the call fails with reason code MQRC_BUFFER_LENGTH_ERROR.
 - The *Buffer* parameter is declared as being of type String. If the data to be placed on the queue is not of type String, the MQPUT1Any call should be used in place of MQPUT1.

The MQPUT1Any call has the same parameters as the MQPUT1 call, except that the *Buffer* parameter is declared as being of type Any, allowing any type of data to be placed on the queue. However, this means that *Buffer* cannot be checked to ensure that it is at least *BufferLength* bytes in size.

- On Compaq NonStop Kernel, if the MQPUT1 call is issued outside a Compaq TMF transaction *without* the MQPMO_NO_SYNCPOINT option, the reason code MQRC_UNIT_OF_WORK_NOT_STARTED is returned.

Language invocations

The MQPUT1 call is supported in the programming languages shown below.

C invocation

```
MQPUT1 (Hconn, &ObjDesc, &MsgDesc, &PutMsgOpts,
        BufferLength, Buffer, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;          /* Connection handle */
MQOD     ObjDesc;       /* Object descriptor */
MQMD     MsgDesc;       /* Message descriptor */
MQPMO    PutMsgOpts;    /* Options that control the action of MQPUT1 */
MQLONG   BufferLength;   /* Length of the message in Buffer */
MQBYTE   Buffer[n];     /* Message data */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;       /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQPUT1' USING HCONN, OBJDESC, MSGDESC, PUTMSGOPTS,
                   BUFFERLENGTH, BUFFER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN          PIC S9(9) BINARY.
** Object descriptor
01 OBJDESC.
   COPY CMQODV.
** Message descriptor
01 MSGDESC.
   COPY CMQMDV.
** Options that control the action of MQPUT1
01 PUTMSGOPTS.
   COPY CMQPMOV.
** Length of the message in BUFFER
01 BUFFERLENGTH PIC S9(9) BINARY.
** Message data
01 BUFFER        PIC X(n).
** Completion code
01 COMPCODE      PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON        PIC S9(9) BINARY.
```

PL/I invocation

```
call MQPUT1 (Hconn, ObjDesc, MsgDesc, PutMsgOpts, BufferLength, Buffer,
             CompCode, Reason);
```

Declare the parameters as follows:

```
dc1 Hconn        fixed bin(31); /* Connection handle */
dc1 ObjDesc      like MQOD;     /* Object descriptor */
dc1 MsgDesc      like MQMD;     /* Message descriptor */
```

MQPUT1 – Language invocations

```
dc1 PutMsgOpts    like MQPMO;    /* Options that control the action of
                               MQPUT1 */
dc1 BufferLength  fixed bin(31); /* Length of the message in Buffer */
dc1 Buffer        char(n);       /* Message data */
dc1 CompCode     fixed bin(31); /* Completion code */
dc1 Reason       fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
CALL MQPUT1,(HCONN,OBJDESC,MSGDESC,PUTMSGOPTS,BUFFERLENGTH, X
            BUFFER,COMPCODE,REASON)
```

Declare the parameters as follows:

HCONN	DS	F	Connection handle
OBJDESC	CMQODA	,	Object descriptor
MSGDESC	CMQMDA	,	Message descriptor
PUTMSGOPTS	CMQPMOA	,	Options that control the action of MQPUT1
BUFFERLENGTH	DS	F	Length of the message in BUFFER
BUFFER	DS	CL(n)	Message data
COMPCODE	DS	F	Completion code
REASON	DS	F	Reason code qualifying COMPCODE

TAL invocation

```
INT(32) .EXT HConn ;
STRUCT  .EXT ObjDesc(MQOD^Def);
STRUCT  .EXT MsgDesc(MQMD^Def);
STRUCT  .EXT PutMsgOpt(MQPMO^Def);
INT(32) .EXT BufferLen
STRING  .EXT Buffer[0:BUFFER^SIZE]
INT(32) .EXT CC;
INT(32) .EXT Reason;

CALL MQPUT1(HConn, ObjDesc, MsgDesc, PutMsgOpt, BufferLen, Buffer,
            CC, Reason);
```

Visual Basic invocation

```
MQPUT1 Hconn, ObjDesc, MsgDesc, PutMsgOpts, BufferLength, Buffer,
      CompCode, Reason
```

Declare the parameters as follows:

Dim Hconn	As Long	'Connection handle'
Dim ObjDesc	As MQOD	'Object descriptor'
Dim MsgDesc	As MQMD	'Message descriptor'
Dim PutMsgOpts	As MQPMO	'Options that control the action of MQPUT1'
Dim BufferLength	As Long	'Length of the message in Buffer'
Dim Buffer	As String	'Message data'
Dim CompCode	As Long	'Completion code'
Dim Reason	As Long	'Reason code qualifying CompCode'

Chapter 39. MQSET – Set object attributes

The MQSET call is used to change the attributes of an object represented by a handle. The object must be a queue.

Syntax

MQSET (*Hconn*, *Hobj*, *SelectorCount*, *Selectors*, *IntAttrCount*,
IntAttrs, *CharAttrLength*, *CharAttrs*, *CompCode*, *Reason*)

Parameters

The MQSET call has the following parameters.

Hconn (MQHCONN) – input

Connection handle.

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on OS/400 for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Hobj (MQHOBJ) – input

Object handle.

This handle represents the queue object whose attributes are to be set. The handle was returned by a previous MQOPEN call that specified the MQOO_SET option.

SelectorCount (MQLONG) – input

Count of selectors.

This is the count of selectors that are supplied in the *Selectors* array. It is the number of attributes that are to be set. Zero is a valid value. The maximum number allowed is 256.

Selectors (MQLONG×SelectorCount) – input

Array of attribute selectors.

This is an array of *SelectorCount* attribute selectors; each selector identifies an attribute (integer or character) whose value is to be set.

Each selector must be valid for the type of queue that *Hobj* represents. Only certain MQIA_* and MQCA_* values are allowed; these values are listed below.

MQSET – Selectors parameter

Selectors can be specified in any order. Attribute values that correspond to integer attribute selectors (MQIA_* selectors) must be specified in *IntAttrs* in the same order in which these selectors occur in *Selectors*. Attribute values that correspond to character attribute selectors (MQCA_* selectors) must be specified in *CharAttrs* in the same order in which those selectors occur. MQIA_* selectors can be interleaved with the MQCA_* selectors; only the relative order within each type is important.

It is not an error to specify the same selector more than once; if this is done, the last value specified for a given selector is the one that takes effect.

Notes:

1. The integer and character attribute selectors are allocated within two different ranges; the MQIA_* selectors reside within the range MQIA_FIRST through MQIA_LAST, and the MQCA_* selectors within the range MQCA_FIRST through MQCA_LAST.

For each range, the constants MQIA_LAST_USED and MQCA_LAST_USED define the highest value that the queue manager will accept.

2. If all the MQIA_* selectors occur first, the same element numbers can be used to address corresponding elements in the *Selectors* and *IntAttrs* arrays.
3. If the *SelectorCount* parameter is zero, *Selectors* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler may be null.

The attributes that can be set are listed in the following table. No other attributes can be set using this call. For the MQCA_* attribute selectors, the constant that defines the length in bytes of the string that is required in *CharAttrs* is given in parentheses.

Table 91. MQSET attribute selectors for queues

Selector	Description	Note
MQCA_TRIGGER_DATA	Trigger data (MQ_TRIGGER_DATA_LENGTH).	2
MQIA_DIST_LISTS	Distribution list support.	1
MQIA_INHIBIT_GET	Whether get operations are allowed.	–
MQIA_INHIBIT_PUT	Whether put operations are allowed.	–
MQIA_TRIGGER_CONTROL	Trigger control.	2
MQIA_TRIGGER_DEPTH	Trigger depth.	2
MQIA_TRIGGER_MSG_PRIORITY	Threshold message priority for triggers.	2
MQIA_TRIGGER_TYPE	Trigger type.	2

Notes:

1. Supported only on AIX, HP-UX, OS/2, Compaq NonStop Kernel, Compaq OpenVMS Alpha, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.
2. Not supported on VSE/ESA.

IntAttrCount (MQLONG) – input

Count of integer attributes.

This is the number of elements in the *IntAttrs* array, and must be at least the number of MQIA_* selectors in the *Selectors* parameter. Zero is a valid value if there are none.

IntAttrs (MQLONG×IntAttrCount) – input

Array of integer attributes.

This is an array of *IntAttrCount* integer attribute values. These attribute values must be in the same order as the MQIA_* selectors in the *Selectors* array.

If the *IntAttrCount* or *SelectorCount* parameter is zero, *IntAttrs* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler may be null.

CharAttrLength (MQLONG) – input

Length of character attributes buffer.

This is the length in bytes of the *CharAttrs* parameter, and must be at least the sum of the lengths of the character attributes specified in the *Selectors* array. Zero is a valid value if there are no MQCA_* selectors in *Selectors*.

CharAttrs (MQCHAR×CharAttrLength) – input

Character attributes.

This is the buffer containing the character attribute values, concatenated together. The length of the buffer is given by the *CharAttrLength* parameter.

The characters attributes must be specified in the same order as the MQCA_* selectors in the *Selectors* array. The length of each character attribute is fixed (see *Selectors*). If the value to be set for an attribute contains fewer nonblank characters than the defined length of the attribute, the value in *CharAttrs* must be padded to the right with blanks to make the attribute value match the defined length of the attribute.

If the *CharAttrLength* or *SelectorCount* parameter is zero, *CharAttrs* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler may be null.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE

(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR

(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR

(2374, X'946') API exit failed.

|
|

MQSET – Reason parameter

MQRC_API_EXIT_LOAD_ERROR	(2183, X'887') Unable to load API exit.
MQRC_ASID_MISMATCH	(2157, X'86D') Primary and home ASIDs differ.
MQRC_CALL_IN_PROGRESS	(2219, X'8AB') MQI call entered before previous call complete.
MQRC_CF_STRUC_FAILED	(2373, X'945') Coupling-facility structure failed.
MQRC_CF_STRUC_IN_USE	(2346, X'92A') Coupling-facility structure in use.
MQRC_CF_STRUC_LIST_HDR_IN_USE	(2347, X'92B') Coupling-facility structure list-header in use.
MQRC_CHAR_ATTR_LENGTH_ERROR	(2006, X'7D6') Length of character attributes not valid.
MQRC_CHAR_ATTRS_ERROR	(2007, X'7D7') Character attributes string not valid.
MQRC_CICS_WAIT_FAILED	(2140, X'85C') Wait request rejected by CICS.
MQRC_CONNECTION_BROKEN	(2009, X'7D9') Connection to queue manager lost.
MQRC_CONNECTION_NOT_AUTHORIZED	(2217, X'8A9') Not authorized for connection.
MQRC_CONNECTION_STOPPING	(2203, X'89B') Connection shutting down.
MQRC_DB2_NOT_AVAILABLE	(2342, X'926') DB2 subsystem not available.
MQRC_HCONN_ERROR	(2018, X'7E2') Connection handle not valid.
MQRC_HOBJ_ERROR	(2019, X'7E3') Object handle not valid.
MQRC_INHIBIT_VALUE_ERROR	(2020, X'7E4') Value for inhibit-get or inhibit-put queue attribute not valid.
MQRC_INT_ATTR_COUNT_ERROR	(2021, X'7E5') Count of integer attributes not valid.
MQRC_INT_ATTRS_ARRAY_ERROR	(2023, X'7E7') Integer attributes array not valid.
MQRC_NOT_OPEN_FOR_SET	(2040, X'7F8') Queue not open for set.
MQRC_OBJECT_CHANGED	(2041, X'7F9') Object definition changed since opened.
MQRC_OBJECT_DAMAGED	(2101, X'835') Object damaged.
MQRC_PAGESET_ERROR	(2193, X'891') Error accessing page-set data set.
MQRC_Q_DELETED	(2052, X'804') Queue has been deleted.
MQRC_Q_MGR_NAME_ERROR	(2058, X'80A') Queue manager name not valid or not known.
MQRC_Q_MGR_NOT_AVAILABLE	(2059, X'80B') Queue manager not available for connection.
MQRC_Q_MGR_STOPPING	(2162, X'872') Queue manager shutting down.
MQRC_RESOURCE_PROBLEM	(2102, X'836') Insufficient system resources available.
MQRC_SELECTOR_COUNT_ERROR	(2065, X'811') Count of selectors not valid.

MQRC_SELECTOR_ERROR	(2067, X'813')	Attribute selector not valid.
MQRC_SELECTOR_LIMIT_EXCEEDED	(2066, X'812')	Count of selectors too big.
MQRC_STORAGE_NOT_AVAILABLE	(2071, X'817')	Insufficient storage available.
MQRC_SUPPRESSED_BY_EXIT	(2109, X'83D')	Call suppressed by exit program.
MQRC_TRIGGER_CONTROL_ERROR	(2075, X'81B')	Value for trigger-control attribute not valid.
MQRC_TRIGGER_DEPTH_ERROR	(2076, X'81C')	Value for trigger-depth attribute not valid.
MQRC_TRIGGER_MSG_PRIORITY_ERR	(2077, X'81D')	Value for trigger-message-priority attribute not valid.
MQRC_TRIGGER_TYPE_ERROR	(2078, X'81E')	Value for trigger-type attribute not valid.
MQRC_UNEXPECTED_ERROR	(2195, X'893')	Unexpected error occurred.

For more information on these reason codes, see Appendix A, “Return codes”, on page 527.

Usage notes

- Using this call, the application can specify an array of integer attributes, or a collection of character attribute strings, or both. The attributes specified are all set simultaneously, if no errors occur. If an error does occur (for example, if a selector is not valid, or an attempt is made to set an attribute to a value that is not valid), the call fails and no attributes are set.
- The values of attributes can be determined using the MQINQ call; see Chapter 35, “MQINQ – Inquire object attributes”, on page 393 for details.

Note: Not all attributes whose values can be inquired using the MQINQ call can have their values changed using the MQSET call. For example, no process-object or queue-manager attributes can be set with this call.
- Attribute changes are preserved across restarts of the queue manager (other than alterations to temporary dynamic queues, which do not survive restarts of the queue manager).
- It is not possible to change the attributes of a model queue using the MQSET call. However, if you open a model queue using the MQOPEN call with the MQOO_SET option, you can use the MQSET call to set the attributes of the dynamic local queue that is created by the MQOPEN call.
- If the object being set is a cluster queue, there must be a local instance of the cluster queue for the open to succeed.
- Changes to attributes resulting from use of the MQSET call do not affect the values of the *AlterationDate* and *AlterationTime* attributes.
- On Compaq NonStop Kernel, for a FASTPATH application opening or closing a dynamic queue, MQSeries can start and end a TM/MP transaction in order to update audited databases. If the application has opened the TM/MP T-file (because it can initiate multiple transactions), ENDTRANSACTION is a no-waited operation, and the application receives a completion for the transaction initiated by MQSeries. Review the design of your applications to determine if this is the case and verify that the logic handling completions can cope with ENDTRANSACTION completions that are caused by MQSeries.

MQSET – Usage notes

8. For more information about object attributes, see:
 - Chapter 40, “Attributes for queues”, on page 457
 - Chapter 41, “Attributes for namelists”, on page 491
 - Chapter 42, “Attributes for process definitions”, on page 495
 - Chapter 43, “Attributes for the queue manager”, on page 501

Language invocations

The MQSET call is supported in the programming languages shown below.

C invocation

```
MQSET (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,  
CharAttrLength, CharAttrs, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;          /* Connection handle */  
MQHOBJ  Hobj;          /* Object handle */  
MQLONG  SelectorCount; /* Count of selectors */  
MQLONG  Selectors[n];  /* Array of attribute selectors */  
MQLONG  IntAttrCount;  /* Count of integer attributes */  
MQLONG  IntAttrs[n];   /* Array of integer attributes */  
MQLONG  CharAttrLength; /* Length of character attributes buffer */  
MQCHAR  CharAttrs[n];  /* Character attributes */  
MQLONG  CompCode;      /* Completion code */  
MQLONG  Reason;        /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQSET' USING HCONN, HOBJ, SELECTORCOUNT, SELECTORS-TABLE,  
INTATTRCOUNT, INTATTRS-TABLE, CHARATTRLENGTH,  
CHARATTRS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle  
01 HCONN          PIC S9(9) BINARY.  
** Object handle  
01 HOBJ          PIC S9(9) BINARY.  
** Count of selectors  
01 SELECTORCOUNT PIC S9(9) BINARY.  
** Array of attribute selectors  
01 SELECTORS-TABLE.  
02 SELECTORS     PIC S9(9) BINARY OCCURS n TIMES.  
** Count of integer attributes  
01 INTATTRCOUNT PIC S9(9) BINARY.  
** Array of integer attributes  
01 INTATTRS-TABLE.  
02 INTATTRS     PIC S9(9) BINARY OCCURS n TIMES.  
** Length of character attributes buffer  
01 CHARATTRLENGTH PIC S9(9) BINARY.  
** Character attributes  
01 CHARATTRS     PIC X(n).  
** Completion code  
01 COMPCODE      PIC S9(9) BINARY.  
** Reason code qualifying COMPCODE  
01 REASON        PIC S9(9) BINARY.
```

PL/I invocation

```
call MQSET (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount,  
IntAttrs, CharAttrLength, CharAttrs, CompCode, Reason);
```

Declare the parameters as follows:

MQSET – Language invocations

```
dc1 Hconn          fixed bin(31); /* Connection handle */
dc1 Hobj           fixed bin(31); /* Object handle */
dc1 SelectorCount  fixed bin(31); /* Count of selectors */
dc1 Selectors(n)   fixed bin(31); /* Array of attribute selectors */
dc1 IntAttrCount   fixed bin(31); /* Count of integer attributes */
dc1 IntAttrs(n)    fixed bin(31); /* Array of integer attributes */
dc1 CharAttrLength fixed bin(31); /* Length of character attributes
                                buffer */

dc1 CharAttrs      char(n);      /* Character attributes */
dc1 CompCode       fixed bin(31); /* Completion code */
dc1 Reason         fixed bin(31); /* Reason code qualifying
                                CompCode */
```

System/390 assembler invocation

```
CALL MQSET,(HCONN,HOBJ,SELECTORCOUNT,SELECTORS,INTATTRCOUNT, X
INTATTRS,CHARATTRLENGTH,CHARATTRS,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN          DS  F      Connection handle
HOBJ           DS  F      Object handle
SELECTORCOUNT DS  F      Count of selectors
SELECTORS      DS  (n)F   Array of attribute selectors
INTATTRCOUNT  DS  F      Count of integer attributes
INTATTRS       DS  (n)F   Array of integer attributes
CHARATTRLENGTH DS  F      Length of character attributes buffer
CHARATTRS      DS  CL(n)  Character attributes
COMPCODE       DS  F      Completion code
REASON         DS  F      Reason code qualifying COMPCODE
```

TAL invocation

```
INT(32) .EXT HConn ;
INT(32) .EXT Hobj;
INT(32) SelectorCount;
INT(32) .EXT Selectors[0:NUM^SELECTORS];
INT(32) IntAttrCount;
INT(32) .EXT IntAttrs[0:NUM^INT^ATTR];
INT(32) CharAttrLength;
STRING .EXT CharAttrs[0:LEN^CHAR^ATTR];
INT(32) .EXT CC;
INT(32) .EXT Reason;
```

```
CALL MQSET(HConn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,
CharAttrLength, CharAttrs, CC, Reason);
```

Visual Basic invocation

```
MQSET Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,
CharAttrLength, CharAttrs, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn          As Long  'Connection handle'
Dim Hobj           As Long  'Object handle'
Dim SelectorCount  As Long  'Count of selectors'
Dim Selectors      As Long  'Array of attribute selectors'
Dim IntAttrCount   As Long  'Count of integer attributes'
Dim IntAttrs       As Long  'Array of integer attributes'
Dim CharAttrLength As Long  'Length of character attributes buffer'
Dim CharAttrs      As String 'Character attributes'
Dim CompCode       As Long  'Completion code'
Dim Reason         As Long  'Reason code qualifying CompCode'
```

MQSET – Language invocations

Part 3. Attributes of objects

Chapter 40. Attributes for queues	457	Attribute descriptions	491
Attribute descriptions	460	AlterationDate (MQCHAR12)	491
AlterationDate (MQCHAR12)	460	AlterationTime (MQCHAR8)	491
AlterationTime (MQCHAR8)	461	NameCount (MQLONG)	492
BackoutRequeueQName (MQCHAR48)	461	NamelistDesc (MQCHAR64)	492
BackoutThreshold (MQLONG)	461	NamelistName (MQCHAR48)	492
BaseQName (MQCHAR48)	462	NamelistType (MQLONG)	493
CFStrucName (MQCHAR12)	462	Names (MQCHAR48×NameCount)	493
ClusterName (MQCHAR48)	462	QSGDisp (MQLONG)	493
ClusterNamelist (MQCHAR48)	463	Chapter 42. Attributes for process definitions	495
CreationDate (MQCHAR12)	463	Attribute descriptions	495
CreationTime (MQCHAR8)	464	AlterationDate (MQCHAR12)	495
CurrentQDepth (MQLONG)	464	AlterationTime (MQCHAR8)	495
DefBind (MQLONG)	464	AppId (MQCHAR256)	496
DefinitionType (MQLONG)	465	AppType (MQLONG)	496
DefInputOpenOption (MQLONG)	466	EnvData (MQCHAR128)	497
DefPersistence (MQLONG)	467	ProcessDesc (MQCHAR64)	498
DefPriority (MQLONG)	468	ProcessName (MQCHAR48)	498
DistLists (MQLONG)	468	QSGDisp (MQLONG)	498
HardenGetBackout (MQLONG)	469	UserData (MQCHAR128)	499
IndexType (MQLONG)	470	Chapter 43. Attributes for the queue manager	501
InhibitGet (MQLONG)	472	Attribute descriptions	502
InhibitPut (MQLONG)	473	AlterationDate (MQCHAR12)	502
InitiationQName (MQCHAR48)	473	AlterationTime (MQCHAR8)	503
MaxMsgLength (MQLONG)	474	AuthorityEvent (MQLONG)	503
MaxQDepth (MQLONG)	475	ChannelAutoDef (MQLONG)	503
MsgDeliverySequence (MQLONG)	475	ChannelAutoDefEvent (MQLONG)	504
OpenInputCount (MQLONG)	476	ChannelAutoDefExit (MQCHARn)	504
OpenOutputCount (MQLONG)	477	ClusterWorkloadData (MQCHAR32)	505
ProcessName (MQCHAR48)	477	ClusterWorkloadExit (MQCHARn)	505
QDepthHighEvent (MQLONG)	477	ClusterWorkloadLength (MQLONG)	505
QDepthHighLimit (MQLONG)	478	CodedCharSetId (MQLONG)	506
QDepthLowEvent (MQLONG)	478	CommandInputQName (MQCHAR48)	506
QDepthLowLimit (MQLONG)	479	CommandLevel (MQLONG)	506
QDepthMaxEvent (MQLONG)	479	DeadLetterQName (MQCHAR48)	508
QDesc (MQCHAR64)	480	DefXmitQName (MQCHAR48)	510
QName (MQCHAR48)	480	DistLists (MQLONG)	510
QServiceInterval (MQLONG)	480	ExpiryInterval (MQLONG)	510
QServiceIntervalEvent (MQLONG)	481	IGQPutAuthority (MQLONG)	510
QSGDisp (MQLONG)	481	IGQUserId (MQLONG)	512
QType (MQLONG)	482	InhibitEvent (MQLONG)	512
RemoteQMgrName (MQCHAR48)	483	IntraGroupQueuing (MQLONG)	512
RemoteQName (MQCHAR48)	483	LocalEvent (MQLONG)	513
RetentionInterval (MQLONG)	484	MaxHandles (MQLONG)	513
Scope (MQLONG)	484	MaxMsgLength (MQLONG)	514
Shareability (MQLONG)	485	MaxPriority (MQLONG)	514
StorageClass (MQCHAR8)	485	MaxUncommittedMsgs (MQLONG)	514
TriggerControl (MQLONG)	486	PerformanceEvent (MQLONG)	515
TriggerData (MQCHAR64)	486	Platform (MQLONG)	516
TriggerDepth (MQLONG)	487	QMgrDesc (MQCHAR64)	516
TriggerMsgPriority (MQLONG)	487	QMgrIdentifier (MQCHAR48)	517
TriggerType (MQLONG)	487	QMgrName (MQCHAR48)	517
Usage (MQLONG)	488	QSGName (MQCHAR4)	517
XmitQName (MQCHAR48)	488	RemoteEvent (MQLONG)	517
Chapter 41. Attributes for namelists	491		

Object attributes

RepositoryName (MQCHAR48)	518
RepositoryNamelist (MQCHAR48)	518
StartStopEvent (MQLONG).	519
SyncPoint (MQLONG)	519
TriggerInterval (MQLONG).	519

Chapter 44. Attributes for authentication	
information objects	521
Attribute descriptions	521
AlterationDate (MQCHAR12)	521
AlterationTime (MQCHAR8)	521
AuthInfoConnName (MQCHAR264).	522
AuthInfoDesc (MQCHAR64)	522
AuthInfoName (MQCHAR48)	522
AuthInfoType (MQLONG)	522
LDAPPassword (MQCHAR32)	522
LDAPUserName	
(MQ_DISTINGUISHED_NAME_LENGTH)	522

Chapter 40. Attributes for queues

Types of queue: The queue manager supports the following types of queue definition:

Local queue

This is a physical queue that stores messages. The queue exists on the local queue manager.

Applications connected to the local queue manager can place messages on and remove messages from queues of this type. The value of the *QType* queue attribute is MQQT_LOCAL.

Shared queue

This is a physical queue that stores messages. The queue exists in a shared repository that is accessible to all of the queue managers that belong to the queue-sharing group that owns the shared repository.

Applications connected to any queue manager in the queue-sharing group can place messages on and remove messages from queues of this type. Such queues are effectively the same as local queues. The value of the *QType* queue attribute is MQQT_LOCAL.

- Shared queues are supported only on z/OS.

Cluster queue

This is a physical queue that stores messages. The queue exists either on the local queue manager, or on one or more of the queue managers that belong to the same cluster as the local queue manager.

Applications connected to the local queue manager can place messages on queues of this type, regardless of the location of the queue. If an instance of the queue exists on the local queue manager, the queue behaves in the same way as a local queue, and applications connected to the local queue manager can remove messages from the queue. The value of the *QType* queue attribute is MQQT_CLUSTER.

Cluster queues are not supported in WebSphere Application Server embedded messaging using reduced function WebSphere MQ.

Remote queue

This is not a physical queue – it is the local definition of a queue that exists on a remote queue manager. The local definition of the remote queue contains information that tells the local queue manager how to route messages to the remote queue manager.

Applications connected to the local queue manager can place messages on queues of this type – the messages are actually placed on the the local transmission queue used to route messages to the remote queue manager. Applications cannot remove messages from remote queues. The value of the *QType* queue attribute is MQQT_REMOTE.

A remote queue definition can also be used for:

- Reply-queue aliasing
In this case the name of the definition is the name of a reply-to queue. For more information, see the *WebSphere MQ Intercommunication* book.
- Queue-manager aliasing

Attributes – queues

In this case the name of the definition is an alias for a queue manager, and not the name of a queue. For more information, see the *WebSphere MQ Intercommunication* book.

Alias queue

This is not a physical queue – it is an alternative name for a local queue, a shared queue, a cluster queue, or a remote queue. The name of the queue to which the alias resolves is part of the definition of the alias queue.

Applications connected to the local queue manager can place messages on queues of this type – the messages are actually placed on the queue to which the alias resolves. Applications can remove messages from queues of this type if the alias resolves to a local queue, a shared queue, or a cluster queue that has a local instance. The value of the *QType* queue attribute is MQQT_ALIAS.

Model queue

This is not a physical queue – it is a set of queue attributes from which a local queue can be created.

Messages cannot be stored on queues of this type.

Queue attributes: Some queue attributes apply to all types of queue; other queue attributes apply only to certain types of queue. The types of queue to which an attribute applies are shown in Table 92 on page 459 and subsequent tables.

Table 92 on page 459 summarizes the attributes that are specific to queues. The attributes are described in alphabetic order.

Note: The names of the attributes shown in this book are the names used with the MQINQ and MQSET calls. When MQSC commands are used to define, alter, or display attributes, alternative short names are used; see the *WebSphere MQ Script (MQSC) Command Reference* for details.

Table 92. Attributes for queues. The columns apply as follows:

- The column for model queues indicates which attributes are inherited by the local queue created from the model queue.
- The column for cluster queues indicates the attributes that can be inquired when the cluster queue is opened for inquire alone, or for inquire and output. If the cluster queue is opened for inquire plus one or more of input, browse, or set, the column for local queues applies instead.

Attribute	Description	Local	Model	Alias	Remote	Cluster	Page
<i>AlterationDate</i>	Date when definition was last changed	Yes	No	Yes	Yes	No	460
<i>AlterationTime</i>	Time when definition was last changed	Yes	No	Yes	Yes	No	461
<i>BackoutRequeueQName</i>	Excessive backout requeue queue name	Yes	Yes	No	No	No	461
<i>BackoutThreshold</i>	Backout threshold	Yes	Yes	No	No	No	461
<i>BaseQName</i>	Queue name to which alias resolves	No	No	Yes	No	No	462
<i>CFStrucName</i>	Coupling-facility structure name	Yes	Yes	No	No	No	462
<i>ClusterName</i>	Name of cluster to which queue belongs	Yes	No	Yes	Yes	No	462
<i>ClusterNameList</i>	Name of namelist object containing names of clusters to which queue belongs	Yes	No	Yes	Yes	No	463
<i>CreationDate</i>	Date the queue was created	Yes	No	No	No	No	463
<i>CreationTime</i>	Time the queue was created	Yes	No	No	No	No	464
<i>CurrentQDepth</i>	Current queue depth	Yes	No	No	No	No	464
<i>DefBind</i>	Default binding	Yes	No	Yes	Yes	Yes	464
<i>DefinitionType</i>	Queue definition type	Yes	Yes	No	No	No	465
<i>DefInputOpenOption</i>	Default input open option	Yes	Yes	No	No	No	466
<i>DefPersistence</i>	Default message persistence	Yes	Yes	Yes	Yes	Yes	467
<i>DefPriority</i>	Default message priority	Yes	Yes	Yes	Yes	Yes	468
<i>DistLists</i>	Distribution list support	Yes	Yes	No	No	No	468
<i>HardenGetBackout</i>	Whether to maintain an accurate backout count	Yes	Yes	No	No	No	469
<i>IndexType</i>	Index type	Yes	Yes	No	No	No	470
<i>InhibitGet</i>	Controls whether get operations for the queue are allowed	Yes	Yes	Yes	No	No	472
<i>InhibitPut</i>	Controls whether put operations for the queue are allowed	Yes	Yes	Yes	Yes	Yes	473
<i>InitiationQName</i>	Name of initiation queue	Yes	Yes	No	No	No	473
<i>MaxMsgLength</i>	Maximum message length in bytes	Yes	Yes	No	No	No	474
<i>MaxQDepth</i>	Maximum queue depth	Yes	Yes	No	No	No	475
<i>MsgDeliverySequence</i>	Message delivery sequence	Yes	Yes	No	No	No	475
<i>OpenInputCount</i>	Number of opens for input	Yes	No	No	No	No	476
<i>OpenOutputCount</i>	Number of opens for output	Yes	No	No	No	No	477
<i>ProcessName</i>	Process name	Yes	Yes	No	No	No	477
<i>QDepthHighEvent</i>	Controls whether Queue Depth High events are generated	Yes	Yes	No	No	No	477
<i>QDepthHighLimit</i>	High limit for queue depth	Yes	Yes	No	No	No	478
<i>QDepthLowEvent</i>	Controls whether Queue Depth Low events are generated	Yes	Yes	No	No	No	478
<i>QDepthLowLimit</i>	Low limit for queue depth	Yes	Yes	No	No	No	479
<i>QDepthMaxEvent</i>	Controls whether Queue Full events are generated	Yes	Yes	No	No	No	479
<i>QDesc</i>	Queue description	Yes	Yes	Yes	Yes	Yes	480
<i>QName</i>	Queue name	Yes	No	Yes	Yes	Yes	480
<i>QServiceInterval</i>	Target for queue service interval	Yes	Yes	No	No	No	480
<i>QServiceIntervalEvent</i>	Controls whether Service Interval High or Service Interval OK events are generated	Yes	Yes	No	No	No	481
<i>QSGDisp</i>	Queue-sharing group disposition	Yes	No	Yes	Yes	Yes	481
<i>QType</i>	Queue type	Yes	No	Yes	Yes	Yes	482

Attributes – queues

Table 92. Attributes for queues (continued). The columns apply as follows:

- The column for model queues indicates which attributes are inherited by the local queue created from the model queue.
- The column for cluster queues indicates the attributes that can be inquired when the cluster queue is opened for inquire alone, or for inquire and output. If the cluster queue is opened for inquire plus one or more of input, browse, or set, the column for local queues applies instead.

Attribute	Description	Local	Model	Alias	Remote	Cluster	Page
<i>RemoteQMgrName</i>	Name of remote queue manager	No	No	No	Yes	No	483
<i>RemoteQName</i>	Name of remote queue	No	No	No	Yes	No	483
<i>RetentionInterval</i>	Retention interval	Yes	Yes	No	No	No	484
<i>Scope</i>	Controls whether an entry for the queue also exists in a cell directory	Yes	No	Yes	Yes	No	484
<i>Shareability</i>	Queue shareability	Yes	Yes	No	No	No	485
<i>StorageClass</i>	Storage class for queue	Yes	Yes	No	No	No	485
<i>TriggerControl</i>	Trigger control	Yes	Yes	No	No	No	486
<i>TriggerData</i>	Trigger data	Yes	Yes	No	No	No	486
<i>TriggerDepth</i>	Trigger depth	Yes	Yes	No	No	No	487
<i>TriggerMsgPriority</i>	Threshold message priority for triggers	Yes	Yes	No	No	No	487
<i>TriggerType</i>	Trigger type	Yes	Yes	No	No	No	487
<i>Usage</i>	Queue usage	Yes	Yes	No	No	No	488
<i>XmitQName</i>	Transmission queue name	No	No	No	Yes	No	488

Attribute descriptions

A queue object has the attributes described below.

AlterationDate (MQCHAR12)

Date when definition was last changed.

Local	Model	Alias	Remote	Cluster
Yes	No	Yes	Yes	No

This is the date when the definition was last changed. The format of the date is YYYY-MM-DD, padded with two trailing blanks to make the length 12 bytes (for example, 1992-09-23bb, where bb represents 2 blank characters).

It is normal for the values of certain attributes to change as the queue manager operates (for example, *CurrentQDepth*). Changes to these attributes do not affect *AlterationDate*. Also, changes resulting from use of the MQSET call do not affect *AlterationDate*.

To determine the value of this attribute, use the MQCA_ALTERATION_DATE selector with the MQINQ call. The length of this attribute is given by MQ_DATE_LENGTH.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems. In MQSeries for Compaq NonStop Kernel, V5.1, the *AlterationDate* attribute is updated only when administrative changes are made to attributes of the queue.

AlterationTime (MQCHAR8)

Time when definition was last changed.

Local	Model	Alias	Remote	Cluster
Yes	No	Yes	Yes	No

This is the time when the definition was last changed. The format of the time is HH.MM.SS using the 24-hour clock, with a leading zero if the hour is less than 10 (for example 09.10.20).

- On z/OS, the time is Greenwich Mean Time (GMT), subject to the system clock being set accurately to GMT.
- In other environments, the time is local time.

It is normal for the values of certain attributes to change as the queue manager operates (for example, *CurrentQDepth*). Changes to these attributes do not affect *AlterationTime*. Also, changes resulting from use of the MQSET call do not affect *AlterationTime*.

To determine the value of this attribute, use the MQCA_ALTERATION_TIME selector with the MQINQ call. The length of this attribute is given by MQ_TIME_LENGTH.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems. In MQSeries for Compaq NonStop Kernel, V5.1, the AlterationTime attribute is updated only when administrative changes are made to attributes of the queue.

BackoutRequeueQName (MQCHAR48)

Excessive backout requeue queue name.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

Apart from allowing its value to be queried, the queue manager takes no action based on the value of this attribute.

To determine the value of this attribute, use the MQCA_BACKOUT_REQ_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

BackoutThreshold (MQLONG)

Backout threshold.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

Apart from allowing its value to be queried, the queue manager takes no action based on the value of this attribute.

To determine the value of this attribute, use the MQIA_BACKOUT_THRESHOLD selector with the MQINQ call.

Queue – BaseQName attribute

BaseQName (MQCHAR48)

The queue name to which the alias resolves.

Local	Model	Alias	Remote	Cluster
No	No	Yes	No	No

This is the name of a queue that is defined to the local queue manager. (For more information on queue names, see the description of the *ObjectName* field in MQOD. The queue is one of the following types:

MQQT_LOCAL

Local queue.

MQQT_REMOTE

Local definition of a remote queue.

MQQT_CLUSTER

Cluster queue.

To determine the value of this attribute, use the MQCA_BASE_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

CFStrucName (MQCHAR12)

Coupling-facility structure name.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This is the name of the coupling-facility structure where the messages on the queue are stored. The first character of the name is in the range A through Z, and the remaining characters are in the range A through Z, 0 through 9, or blank.

The full name of the structure in the coupling facility is obtained by suffixing the value of the *QSGName* queue-manager attribute with the value of the *CFStrucName* queue attribute.

This attribute applies only to shared queues; it is ignored if *QSGDisp* does not have the value MQQSGD_SHARED.

To determine the value of this attribute, use the MQCA_CF_STRUC_NAME selector with the MQINQ call. The length of this attribute is given by MQ_CF_STRUC_NAME_LENGTH.

This attribute is supported only on z/OS; it is ignored if not in a queue-sharing group.

ClusterName (MQCHAR48)

Name of cluster to which queue belongs.

Local	Model	Alias	Remote	Cluster
Yes	No	Yes	Yes	No

Queue – ClusterName attribute

This is the name of the cluster to which the queue belongs. If the queue belongs to more than one cluster, *ClusterNameList* specifies the name of a namelist object that identifies the clusters, and *ClusterName* is blank. At least one of *ClusterName* and *ClusterNameList* must be blank.

To determine the value of this attribute, use the MQCA_CLUSTER_NAME selector with the MQINQ call. The length of this attribute is given by MQ_CLUSTER_NAME_LENGTH.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems. It must be blank in WebSphere Application Server embedded messaging using reduced function WebSphere MQ.

ClusterNameList (MQCHAR48)

Name of namelist object containing names of clusters to which queue belongs.

Local	Model	Alias	Remote	Cluster
Yes	No	Yes	Yes	No

This is the name of a namelist object that contains the names of clusters to which this queue belongs. If the queue belongs to only one cluster, the namelist object contains only one name. Alternatively, *ClusterName* can be used to specify the name of the cluster, in which case *ClusterNameList* is blank. At least one of *ClusterName* and *ClusterNameList* must be blank.

To determine the value of this attribute, use the MQCA_CLUSTER_NAMELIST selector with the MQINQ call. The length of this attribute is given by MQ_NAMELIST_NAME_LENGTH.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems. It must be blank in WebSphere Application Server embedded messaging using reduced function WebSphere MQ.

CreationDate (MQCHAR12)

Date when queue was created.

Local	Model	Alias	Remote	Cluster
Yes	No	No	No	No

This is the date when the queue was created. The format of the date is YYYY-MM-DD, padded with two trailing blanks to make the length 12 bytes (for example, 1992-09-23**bb**, where **bb** represents 2 blank characters).

- On OS/400, the creation date of a queue may differ from that of the underlying operating system entity (file or userspace) that represents the queue.

To determine the value of this attribute, use the MQCA_CREATION_DATE selector with the MQINQ call. The length of this attribute is given by MQ_CREATION_DATE_LENGTH.

Queue – CreationTime attribute

CreationTime (MQCHAR8)

Time when queue was created.

Local	Model	Alias	Remote	Cluster
Yes	No	No	No	No

This is the time when the queue was created. The format of the time is HH.MM.SS using the 24-hour clock, with a leading zero if the hour is less than 10 (for example 09.10.20).

- On z/OS, the time is Greenwich Mean Time (GMT), subject to the system clock being set accurately to GMT.
- In other environments, the time is local time.
- On OS/400, the creation time of a queue may differ from that of the underlying operating system entity (file or userspace) that represents the queue.

To determine the value of this attribute, use the MQCA_CREATION_TIME selector with the MQINQ call. The length of this attribute is given by MQ_CREATION_TIME_LENGTH.

CurrentQDepth (MQLONG)

Current queue depth.

Local	Model	Alias	Remote	Cluster
Yes	No	No	No	No

This is the number of messages currently on the queue. It is incremented during an MQPUT call, and during backout of an MQGET call. It is decremented during a nonbrowse MQGET call, and during backout of an MQPUT call. The effect of this is that the count includes messages that have been put on the queue within a unit of work, but which have not yet been committed, even though they are not eligible to be retrieved by the MQGET call. Similarly, it excludes messages that have been retrieved within a unit of work using the MQGET call, but which have yet to be committed.

The count also includes messages which have passed their expiry time but have not yet been discarded, although these messages are not eligible to be retrieved. See the *Expiry* field described in Chapter 10, “MQMD – Message descriptor”, on page 141.

Unit-of-work processing and the segmentation of messages can both cause *CurrentQDepth* to exceed *MaxQDepth*. However, this does not affect the retrievability of the messages – *all* messages on the queue can be retrieved using the MQGET call in the normal way.

The value of this attribute fluctuates as the queue manager operates.

To determine the value of this attribute, use the MQIA_CURRENT_Q_DEPTH selector with the MQINQ call.

DefBind (MQLONG)

Default binding.

Local	Model	Alias	Remote	Cluster
Yes	No	Yes	Yes	Yes

This is the default binding that is used when MQOO_BIND_AS_Q_DEF is specified on the MQOPEN call and the queue is a cluster queue. The value is one of the following:

MQBND_BIND_ON_OPEN

Binding fixed by MQOPEN call.

MQBND_BIND_NOT_FIXED

Binding not fixed.

To determine the value of this attribute, use the MQIA_DEF_BIND selector with the MQINQ call.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

DefinitionType (MQLONG)

Queue definition type.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This indicates how the queue was defined. The value is one of the following:

MQQDT_PREDEFINED

Predefined permanent queue.

The queue is a permanent queue created by the system administrator; only the system administrator can delete it.

Predefined queues are created using the DEFINE MQSC command, and can be deleted only by using the DELETE MQSC command. Predefined queues cannot be created from model queues.

Commands can be issued either by an operator, or by an authorized user sending a command message to the command input queue (see the *CommandInputQName* attribute described in Chapter 43, “Attributes for the queue manager”, on page 501).

MQQDT_PERMANENT_DYNAMIC

Dynamically defined permanent queue.

The queue is a permanent queue that was created by an application issuing an MQOPEN call with the name of a model queue specified in the object descriptor MQOD. The model queue definition had the value MQQDT_PERMANENT_DYNAMIC for the *DefinitionType* attribute.

This type of queue can be deleted using the MQCLOSE call. See Chapter 29, “MQCLOSE – Close object”, on page 345 for more details.

The value of the *QSGDisp* attribute for a permanent dynamic queue is MQQSGD_Q_MGR.

MQQDT_TEMPORARY_DYNAMIC

Dynamically defined temporary queue.

Queue – DefinitionType attribute

The queue is a temporary queue that was created by an application issuing an MQOPEN call with the name of a model queue specified in the object descriptor MQOD. The model queue definition had the value MQQDT_TEMPORARY_DYNAMIC for the *DefinitionType* attribute.

This type of queue is deleted automatically by the MQCLOSE call when it is closed by the application that created it.

The value of the *QSGDisp* attribute for a temporary dynamic queue is MQQSGD_Q_MGR.

MQQDT_SHARED_DYNAMIC

Dynamically defined shared queue.

The queue is a shared permanent queue that was created by an application issuing an MQOPEN call with the name of a model queue specified in the object descriptor MQOD. The model queue definition had the value MQQDT_SHARED_DYNAMIC for the *DefinitionType* attribute.

This type of queue can be deleted using the MQCLOSE call. See Chapter 29, “MQCLOSE – Close object”, on page 345 for more details.

The value of the *QSGDisp* attribute for a shared dynamic queue is MQQSGD_SHARED.

Shared queues are not supported outside a queue-sharing group.

This attribute in a model queue definition does not indicate how the model queue was defined, because model queues are always predefined. Instead, the value of this attribute in the model queue is used to determine the *DefinitionType* of each of the dynamic queues created from the model queue definition using the MQOPEN call.

To determine the value of this attribute, use the MQIA_DEFINITION_TYPE selector with the MQINQ call.

DefInputOpenOption (MQLONG)

Default input open option.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This is the default way in which the queue should be opened for input. It applies if the MQOO_INPUT_AS_Q_DEF option is specified on the MQOPEN call when the queue is opened. The value is one of the following:

MQOO_INPUT_EXCLUSIVE

Open queue to get messages with exclusive access.

The queue is opened for use with subsequent MQGET calls. The call fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open by this or another application for input of any type (MQOO_INPUT_SHARED or MQOO_INPUT_EXCLUSIVE).

MQOO_INPUT_SHARED

Open queue to get messages with shared access.

The queue is opened for use with subsequent MQGET calls. The call can succeed if the queue is currently open by this or another application with

Queue – DefInputOpenOption attribute

MQOO_INPUT_SHARED, but fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open with MQOO_INPUT_EXCLUSIVE.

To determine the value of this attribute, use the MQIA_DEF_INPUT_OPEN_OPTION selector with the MQINQ call.

DefPersistence (MQLONG)

Default message persistence.

Local	Model	Alias	Remote	Cluster
Yes	Yes	Yes	Yes	Yes

This is the default persistence of messages on the queue. It applies if MQPER_PERSISTENCE_AS_Q_DEF is specified in the message descriptor when the message is put.

If there is more than one definition in the queue-name resolution path, the default persistence is taken from the value of this attribute in the *first* definition in the path at the time of the MQPUT or MQPUT1 call. This could be:

- An alias queue
- A local queue
- A local definition of a remote queue
- A queue-manager alias
- A transmission queue (for example, the *DefXmitQName* queue)

The value is one of the following:

MQPER_PERSISTENT

Message is persistent.

This means that the message survives system failures and restarts of the queue manager. Persistent messages cannot be placed on:

- Temporary dynamic queues
- Shared queues that map to a CFSTRUCT object at CFLEVEL(2) or below, or where the CFSTRUCT object is defined as RECOVER(NO).

Persistent messages can be placed on permanent dynamic queues, and predefined queues.

MQPER_NOT_PERSISTENT

Message is not persistent.

This means that the message does not normally survive system failures or restarts of the queue manager. This applies even if an intact copy of the message is found on auxiliary storage during restart of the queue manager.

In the special case of shared queues, nonpersistent messages *do* survive restarts of queue managers in the queue-sharing group, but do not survive failures of the coupling facility used to store messages on the shared queues.

- On VSE/ESA, MQPER_NOT_PERSISTENT is not supported.

Both persistent and nonpersistent messages can exist on the same queue.

To determine the value of this attribute, use the MQIA_DEF_PERSISTENCE selector with the MQINQ call.

Queue – DefPriority attribute

DefPriority (MQLONG)

Default message priority

Local	Model	Alias	Remote	Cluster
Yes	Yes	Yes	Yes	Yes

This is the default priority for messages on the queue. This applies if MQPRI_PRIORITY_AS_Q_DEF is specified in the message descriptor when the message is put on the queue.

If there is more than one definition in the queue-name resolution path, the default priority for the message is taken from the value of this attribute in the *first* definition in the path at the time of the put operation. This could be:

- An alias queue
- A local queue
- A local definition of a remote queue
- A queue-manager alias
- A transmission queue (for example, the *DefXmitQName* queue)

The way in which a message is placed on a queue depends on the value of the queue's *MsgDeliverySequence* attribute:

- If the *MsgDeliverySequence* attribute is MQMDS_PRIORITY, the logical position at which a message is placed on the queue is dependent on the value of the *Priority* field in the message descriptor.
- If the *MsgDeliverySequence* attribute is MQMDS_FIFO, messages are placed on the queue as though they had a priority equal to the *DefPriority* of the resolved queue, regardless of the value of the *Priority* field in the message descriptor. However, the *Priority* field retains the value specified by the application that put the message. See the *MsgDeliverySequence* attribute described in Chapter 40, "Attributes for queues", on page 457 for more information.

Priorities are in the range zero (lowest) through *MaxPriority* (highest); see the *MaxPriority* attribute described in Chapter 43, "Attributes for the queue manager", on page 501.

To determine the value of this attribute, use the MQIA_DEF_PRIORITY selector with the MQINQ call.

DistLists (MQLONG)

Distribution list support.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This indicates whether distribution-list messages can be placed on the queue. The attribute is set by a message channel agent (MCA) to inform the local queue manager whether the queue manager at the other end of the channel supports distribution lists. This latter queue manager (called the "partnering queue manager") is the one which next receives the message, after it has been removed from the local transmission queue by a sending MCA.

The attribute is set by the sending MCA whenever it establishes a connection to the receiving MCA on the partnering queue manager. In this way, the sending

MCA can cause the local queue manager to place on the transmission queue only messages which the partnering queue manager is capable of processing correctly.

This attribute is primarily for use with transmission queues, but the processing described is performed regardless of the usage defined for the queue (see the *Usage* attribute).

The value is one of the following:

MQDL_SUPPORTED

Distribution lists supported.

This indicates that distribution-list messages can be stored on the queue, and transmitted to the partnering queue manager in that form. This reduces the amount of processing required to send the message to multiple destinations.

MQDL_NOT_SUPPORTED

Distribution lists not supported.

This indicates that distribution-list messages cannot be stored on the queue, because the partnering queue manager does not support distribution lists. If an application puts a distribution-list message, and that message is to be placed on this queue, the queue manager splits the distribution-list message and places the individual messages on the queue instead. This increases the amount of processing required to send the message to multiple destinations, but ensures that the messages will be processed correctly by the partnering queue manager.

To determine the value of this attribute, use the MQIA_DIST_LISTS selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

This attribute is supported in the following environments: AIX, HP-UX, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

HardenGetBackout (MQLONG)

Whether to maintain an accurate backout count.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

For each message, a count is kept of the number of times that the message is retrieved by an MQGET call within a unit of work, and that unit of work subsequently backed out. This count is available in the *BackoutCount* field in the message descriptor after the MQGET call has completed.

The message backout count survives restarts of the queue manager. However, to ensure that the count is accurate, information has to be “hardened” (recorded on disk or other permanent storage device) each time a message is retrieved by an MQGET call within a unit of work for this queue. If this is not done, and a failure of the queue manager occurs together with backout of the MQGET call, the count may or may not be incremented.

Hardening information for each MQGET call within a unit of work, however, imposes a performance overhead, and the *HardenGetBackout* attribute should be set to MQQA_BACKOUT_HARDENED only if it is essential that the count is accurate.

Queue – HardenGetBackout attribute

- On Compaq OpenVMS Alpha, OS/2, OS/400, Compaq NonStop Kernel, UNIX systems, and Windows, the message backout count is always hardened, regardless of the setting of this attribute.

The following values are possible:

MQQA_BACKOUT_HARDENED

Backout count remembered.

Hardening is used to ensure that the backout count for messages on this queue is accurate.

MQQA_BACKOUT_NOT_HARDENED

Backout count may not be remembered.

Hardening is not used to ensure that the backout count for messages on this queue is accurate. The count may therefore be lower than it should be.

To determine the value of this attribute, use the MQIA_HARDEN_GET_BACKOUT selector with the MQINQ call.

IndexType (MQLONG)

Index type.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This specifies the type of index that the queue manager maintains for messages on the queue. The type of index required depends on how the messages are retrieved by the application, and whether the queue is a shared queue or a nonshared queue (see the *QSGDisp* attribute). The following values are possible for *IndexType*:

MQIT_NONE

No index.

No index is maintained by the queue manager for this queue. This value should be used for queues that are usually processed sequentially, that is, without using any selection criteria on the MQGET call.

MQIT_MSG_ID

Queue is indexed using message identifiers.

The queue manager maintains an index that uses the message identifiers of the messages on the queue. This value should be used for queues where the application usually retrieves messages using the message identifier as the selection criterion on the MQGET call.

MQIT_CORREL_ID

Queue is indexed using correlation identifiers.

The queue manager maintains an index that uses the correlation identifiers of the messages on the queue. This value should be used for queues where the application usually retrieves messages using the correlation identifier as the selection criterion on the MQGET call.

MQIT_MSG_TOKEN

Queue is indexed using message tokens.

Queue – IndexType attribute

The queue manager maintains an index that uses the message tokens of the messages on the queue. This value *must* be used for queues where the application retrieves messages using the message token as the selection criterion on the MQGET call.

MQIT_GROUP_ID

Queue is indexed using group identifiers.

The queue manager maintains an index that uses the group identifiers of the messages on the queue. This value *must* be used for queues where the application retrieves messages using the MQGMO_LOGICAL_ORDER option on the MQGET call.

A queue with this index type cannot be a transmission queue. A shared queue with this index type must be defined to map to a CFSTRUCT object at CFLEVEL(3).

Notes:

1. The physical order of messages on a queue with index type MQIT_GROUP_ID is not defined, as the queue is optimized for efficient retrieval of messages using the MQGMO_LOGICAL_ORDER option on the MQGET call. This means that the physical order of the messages is not usually the order in which the messages arrived on the queue.
2. If an MQIT_GROUP_ID queue has a *MsgDeliverySequence* of MQMDS_PRIORITY, the queue manager uses message priorities zero and one to optimize the retrieval of messages in logical order. As a result, the first message in a group should not have a priority of zero or one; if it does, the message is processed as though it had a priority of two. The *Priority* field in the MQMD structure is not changed.

For more information about message groups, see the description of the group and segment options under the *Options* field in MQGMO.

The index type that should be used in various cases is shown in Table 93 and Table 94 on page 472.

Table 93. Recommended or required values of queue index type when MQGMO_LOGICAL_ORDER not specified

Selection criteria on MQGET call	Index type for nonshared queue	Index type for shared queue
None	Any	Any
Selection using one identifier:		
Message identifier	MQIT_MSG_ID recommended	MQIT_MSG_ID required
Correlation identifier	MQIT_CORREL_ID recommended	MQIT_CORREL_ID required
Group identifier	MQIT_GROUP_ID recommended	MQIT_GROUP_ID required
Selection using two identifiers:		
Message identifier plus correlation identifier	MQIT_MSG_ID or MQIT_CORREL_ID recommended	MQIT_MSG_ID or MQIT_CORREL_ID required
Message identifier plus group identifier	MQIT_MSG_ID or MQIT_GROUP_ID recommended	Not supported
Correlation identifier plus group identifier	MQIT_CORREL_ID or MQIT_GROUP_ID recommended	Not supported

Queue – IndexType attribute

Table 93. Recommended or required values of queue index type when MQGMO_LOGICAL_ORDER not specified (continued)

Selection criteria on MQGET call	Index type for nonshared queue	Index type for shared queue
Selection using three identifiers:		
Message identifier plus correlation identifier plus group identifier	MQIT_MSG_ID or MQIT_CORREL_ID or MQIT_GROUP_ID recommended	Not supported
Selection using group-related criteria:		
Group identifier plus message sequence number	MQIT_GROUP_ID required	MQIT_GROUP_ID required
Message sequence number (must be 1)	MQIT_GROUP_ID required	MQIT_GROUP_ID required
Selection using message token:		
Message token	MQIT_MSG_TOKEN required	Not supported

Table 94. Recommended or required values of queue index type when MQGMO_LOGICAL_ORDER specified

Selection criteria on MQGET call	Index type for nonshared queue	Index type for shared queue
None	MQIT_GROUP_ID required	MQIT_GROUP_ID required
Selection using one identifier:		
Message identifier	MQIT_GROUP_ID required	Not supported
Correlation identifier	MQIT_GROUP_ID required	Not supported
Group identifier	MQIT_GROUP_ID required	MQIT_GROUP_ID required
Selection using two identifiers:		
Message identifier plus correlation identifier	MQIT_GROUP_ID required	Not supported
Message identifier plus group identifier	MQIT_GROUP_ID required	Not supported
Correlation identifier plus group identifier	MQIT_GROUP_ID required	Not supported
Selection using three identifiers:		
Message identifier plus correlation identifier plus group identifier	MQIT_GROUP_ID required	Not supported

To determine the value of this attribute, use the MQIA_INDEX_TYPE selector with the MQINQ call.

This attribute is supported only on z/OS.

InhibitGet (MQLONG)

Controls whether get operations for this queue are allowed.

Local	Model	Alias	Remote	Cluster
Yes	Yes	Yes	No	No

If the queue is an alias queue, get operations must be allowed for both the alias and the base queue at the time of the get operation, in order for the MQGET call to succeed. The value is one of the following:

MQQA_GET_INHIBITED

Get operations are inhibited.

MQGET calls fail with reason code MQRC_GET_INHIBITED. This includes MQGET calls that specify MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT.

Note: If an MQGET call operating within a unit of work completes successfully, changing the value of the *InhibitGet* attribute subsequently to MQQA_GET_INHIBITED does not prevent the unit of work being committed.

MQQA_GET_ALLOWED

Get operations are allowed.

To determine the value of this attribute, use the MQIA_INHIBIT_GET selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

InhibitPut (MQLONG)

Controls whether put operations for this queue are allowed.

Local	Model	Alias	Remote	Cluster
Yes	Yes	Yes	Yes	Yes

If there is more than one definition in the queue-name resolution path, put operations must be allowed for *every* definition in the path (including any queue-manager alias definitions) at the time of the put operation, in order for the MQPUT or MQPUT1 call to succeed. The value is one of the following:

MQQA_PUT_INHIBITED

Put operations are inhibited.

MQPUT and MQPUT1 calls fail with reason code MQRC_PUT_INHIBITED.

Note: If an MQPUT call operating within a unit of work completes successfully, changing the value of the *InhibitPut* attribute subsequently to MQQA_PUT_INHIBITED does not prevent the unit of work being committed.

MQQA_PUT_ALLOWED

Put operations are allowed.

To determine the value of this attribute, use the MQIA_INHIBIT_PUT selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

InitiationQName (MQCHAR48)

Name of initiation queue.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This is the name of a queue defined on the local queue manager; the queue must be of type MQQT_LOCAL. The queue manager sends a trigger message to the initiation queue when application start-up is required as a result of a message arriving on the queue to which this attribute belongs. The initiation queue must be

Queue – InitiationQName attribute

monitored by a trigger monitor application which will start the appropriate application after receipt of the trigger message.

To determine the value of this attribute, use the MQCA_INITIATION_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

This attribute is not supported in the following environments: Windows 3.1, Windows 95, Windows 98.

MaxMsgLength (MQLONG)

Maximum message length in bytes.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This is an upper limit for the length of the longest *physical* message that can be placed on the queue. However, because the *MaxMsgLength* queue attribute can be set independently of the *MaxMsgLength* queue-manager attribute, the actual upper limit for the length of the longest physical message that can be placed on the queue is the lesser of those two values.

If the queue manager supports segmentation, it is possible for an application to put a *logical* message that is longer than the lesser of the two *MaxMsgLength* attributes, but only if the application specifies the MQMF_SEGMENTATION_ALLOWED flag in MQMD. If that flag is specified, the upper limit for the length of a logical message is 999 999 999 bytes, but usually resource constraints imposed by the operating system, or by the environment in which the application is running, will result in a lower limit.

An attempt to place on the queue a message that is too long fails with one of the following reason codes:

- MQRC_MSG_TOO_BIG_FOR_Q if the message is too big for the queue
- MQRC_MSG_TOO_BIG_FOR_Q_MGR if the message is too big for the queue manager, but not too big for the queue

The lower limit for the *MaxMsgLength* attribute is zero. The upper limit is determined by the environment:

- On z/OS:
 - For shared queues, the maximum message length is 63 KB (64 512 bytes).
 - For nonshared queues, the maximum message length is 100 MB (104 857 600 bytes).
- On AIX, Compaq NonStop Kernel, Compaq OpenVMS Alpha, HP-UX, OS/2, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems, the maximum message length is 100 MB (104 857 600 bytes).
- On UNIX systems not listed above, Windows 3.1, Windows 95, Windows 98, plus WebSphere MQ clients connected to these systems, the maximum message length is 4 MB (4 194 304 bytes).

For more information, see the *BufferLength* parameter described in Chapter 37, “MQPUT – Put message”, on page 423.

Queue – MaxMsgLength attribute

To determine the value of this attribute, use the MQIA_MAX_MSG_LENGTH selector with the MQINQ call.

MaxQDepth (MQLONG)

Maximum queue depth.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This is the defined upper limit for the number of physical messages that can exist on the queue at any one time. An attempt to put a message on a queue that already contains *MaxQDepth* messages fails with reason code MQRC_Q_FULL.

Unit-of-work processing and the segmentation of messages can both cause the actual number of physical messages on the queue to exceed *MaxQDepth*. However, this does not affect the retrievability of the messages – *all* messages on the queue can be retrieved using the MQGET call in the normal way.

The value of this attribute is zero or greater. The upper limit is determined by the environment:

- On AIX, HP-UX, z/OS, Solaris, Linux, and Windows, the value cannot exceed 999 999 999.
- In other environments, the value cannot exceed 640 000.

Note: It is possible for the storage space available to the queue to be exhausted even if there are fewer than *MaxQDepth* messages on the queue.

To determine the value of this attribute, use the MQIA_MAX_Q_DEPTH selector with the MQINQ call.

MsgDeliverySequence (MQLONG)

Message delivery sequence.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This determines the order in which messages are returned to the application by the MQGET call:

MQMDS_FIFO

Messages are returned in FIFO order (first in, first out).

This means that an MQGET call will return the *first* message that satisfies the selection criteria specified on the call, regardless of the priority of the message.

MQMDS_PRIORITY

Messages are returned in priority order.

This means that an MQGET call will return the *highest-priority* message that satisfies the selection criteria specified on the call. Within each priority level, messages are returned in FIFO order (first in, first out).

- On z/OS, if the queue has an *IndexType* of MQIT_GROUP_ID, the *MsgDeliverySequence* attribute specifies the order in which message groups are returned to the application. The particular sequence in which the groups are

Queue – *MsgDeliverySequence* attribute

returned is determined by the position or priority of the first message in each group. The physical order of messages on the queue is not defined, as the queue is optimized for efficient retrieval of messages using the *MQGMO_LOGICAL_ORDER* option on the *MQGET* call.

- On z/OS, if *IndexType* is *MQIT_GROUP_ID* and *MsgDeliverySequence* is *MQMDS_PRIORITY*, the queue manager uses message priorities zero and one to optimize the retrieval of messages in logical order. As a result, the first message in a group should not have a priority of zero or one; if it does, the message is processed as though it had a priority of two. The *Priority* field in the *MQMD* structure is not changed.

If the relevant attributes are changed while there are messages on the queue, the delivery sequence is as follows:

- The order in which messages are returned by the *MQGET* call is determined by the values of the *MsgDeliverySequence* and *DefPriority* attributes in force for the queue at the time that the message arrives on the queue:
 - If *MsgDeliverySequence* is *MQMDS_FIFO* when the message arrives, the message is placed on the queue as though its priority were *DefPriority*. This does not affect the value of the *Priority* field in the message descriptor of the message; that field retains the value it had when the message was first put.
 - If *MsgDeliverySequence* is *MQMDS_PRIORITY* when the message arrives, the message is placed on the queue at the place appropriate to the priority given by the *Priority* field in the message descriptor.

If the value of the *MsgDeliverySequence* attribute is changed while there are messages on the queue, the order of the messages on the queue is not changed.

If the value of the *DefPriority* attribute is changed while there are messages on the queue, the messages will not necessarily be delivered in FIFO order, even though the *MsgDeliverySequence* attribute is set to *MQMDS_FIFO*; those that were placed on the queue at the higher priority are delivered first.

To determine the value of this attribute, use the *MQIA_MSG_DELIVERY_SEQUENCE* selector with the *MQINQ* call.

OpenInputCount (MQLONG)

Number of opens for input.

Local	Model	Alias	Remote	Cluster
Yes	No	No	No	No

This is the number of handles that are currently valid for removing messages from the queue by means of the *MQGET* call. It is the total number of such handles known to the *local* queue manager. If the queue is a shared queue, the count does not include opens for input that were performed for the queue at other queue managers in the queue-sharing group to which the local queue manager belongs.

The count includes handles where an alias queue which resolves to this queue was opened for input. The count does not include handles where the queue was opened for action(s) which did not include input (for example, a queue opened only for browse).

The value of this attribute fluctuates as the queue manager operates.

Queue – OpenInputCount attribute

To determine the value of this attribute, use the MQIA_OPEN_INPUT_COUNT selector with the MQINQ call.

OpenOutputCount (MQLONG)

Number of opens for output.

Local	Model	Alias	Remote	Cluster
Yes	No	No	No	No

This is the number of handles that are currently valid for adding messages to the queue by means of the MQPUT call. It is the total number of such handles known to the *local* queue manager; it does not include opens for output that were performed for this queue at remote queue managers. If the queue is a shared queue, the count does not include opens for output that were performed for the queue at other queue managers in the queue-sharing group to which the local queue manager belongs.

The count includes handles where an alias queue which resolves to this queue was opened for output. The count does not include handles where the queue was opened for action(s) which did not include output (for example, a queue opened only for inquire).

The value of this attribute fluctuates as the queue manager operates.

To determine the value of this attribute, use the MQIA_OPEN_OUTPUT_COUNT selector with the MQINQ call.

ProcessName (MQCHAR48)

Process name.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This is the name of a process object that is defined on the local queue manager. The process object identifies a program that can service the queue.

To determine the value of this attribute, use the MQCA_PROCESS_NAME selector with the MQINQ call. The length of this attribute is given by MQ_PROCESS_NAME_LENGTH.

QDepthHighEvent (MQLONG)

Controls whether Queue Depth High events are generated.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

A Queue Depth High event indicates that an application has put a message on a queue, and this has caused the number of messages on the queue to become greater than or equal to the queue depth high threshold (see the *QDepthHighLimit* attribute).

Note: The value of this attribute can change dynamically.

Queue – QDepthHighEvent attribute

The value is one of the following:

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

For more information about events, see the *WebSphere MQ Event Monitoring* book.

To determine the value of this attribute, use the MQIA_Q_DEPTH_HIGH_EVENT selector with the MQINQ call.

- This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value. This attribute is ignored on WebSphere Application Server embedded messaging using reduced function WebSphere MQ.

QDepthHighLimit (MQLONG)

High limit for queue depth.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This is the threshold against which the queue depth is compared to generate a Queue Depth High event. This event indicates that an application has put a message on a queue, and this has caused the number of messages on the queue to become greater than or equal to the queue depth high threshold. See the *QDepthHighEvent* attribute.

The value is expressed as a percentage of the maximum queue depth (*MaxQDepth* attribute), and is greater than or equal to 0 and less than or equal to 100. The default value is 80.

To determine the value of this attribute, use the MQIA_Q_DEPTH_HIGH_LIMIT selector with the MQINQ call.

- This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value. This attribute is ignored on WebSphere Application Server embedded messaging using reduced function WebSphere MQ.

QDepthLowEvent (MQLONG)

Controls whether Queue Depth Low events are generated.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

A Queue Depth Low event indicates that an application has retrieved a message from a queue, and this has caused the number of messages on the queue to become less than or equal to the queue depth low threshold (see the *QDepthLowLimit* attribute).

Note: The value of this attribute can change dynamically.

The value is one of the following:

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

For more information about events, see the *WebSphere MQ Event Monitoring* book.

To determine the value of this attribute, use the MQIA_Q_DEPTH_LOW_EVENT selector with the MQINQ call.

- On z/OS, the MQINQ call cannot be used to determine the value of this attribute. This attribute is ignored on WebSphere Application Server embedded messaging using reduced function WebSphere MQ.

QDepthLowLimit (MQLONG)

Low limit for queue depth.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This is the threshold against which the queue depth is compared to generate a Queue Depth Low event. This event indicates that an application has retrieved a message from a queue, and this has caused the number of messages on the queue to become less than or equal to the queue depth low threshold. See the *QDepthLowEvent* attribute.

The value is expressed as a percentage of the maximum queue depth (*MaxQDepth* attribute), and is greater than or equal to 0 and less than or equal to 100. The default value is 20.

To determine the value of this attribute, use the MQIA_Q_DEPTH_LOW_LIMIT selector with the MQINQ call.

- This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value. This attribute is ignored on WebSphere Application Server embedded messaging using reduced function WebSphere MQ.

QDepthMaxEvent (MQLONG)

Controls whether Queue Full events are generated.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

A Queue Full event indicates that a put to a queue has been rejected because the queue is full, that is, the queue depth has already reached its maximum value.

Note: The value of this attribute can change dynamically.

The value is one of the following:

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

For more information about events, see the *WebSphere MQ Event Monitoring* book.

Queue – QDepthMaxEvent attribute

To determine the value of this attribute, use the MQIA_Q_DEPTH_MAX_EVENT selector with the MQINQ call.

- On z/OS, the MQINQ call cannot be used to determine the value of this attribute. This attribute is ignored on WebSphere Application Server embedded messaging using reduced function WebSphere MQ.

QDesc (MQCHAR64)

Queue description.

Local	Model	Alias	Remote	Cluster
Yes	Yes	Yes	Yes	Yes

This is a field that may be used for descriptive commentary. The content of the field is of no significance to the queue manager, but the queue manager may require that the field contain only characters that can be displayed. It cannot contain any null characters; if necessary, it is padded to the right with blanks. In a DBCS installation, the field can contain DBCS characters (subject to a maximum field length of 64 bytes).

Note: If this field contains characters that are not in the queue manager's character set (as defined by the *CodedCharSetId* queue manager attribute), those characters may be translated incorrectly if this field is sent to another queue manager.

To determine the value of this attribute, use the MQCA_Q_DESC selector with the MQINQ call. The length of this attribute is given by MQ_Q_DESC_LENGTH.

QName (MQCHAR48)

Queue name.

Local	Model	Alias	Remote	Cluster
Yes	No	Yes	Yes	Yes

This is the name of a queue defined on the local queue manager. For more information about queue names, see the *WebSphere MQ Application Programming Guide*. All queues defined on a queue manager share the same queue name space. Therefore, a MQQT_LOCAL queue and a MQQT_ALIAS queue cannot have the same name.

To determine the value of this attribute, use the MQCA_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

QServiceInterval (MQLONG)

Target for queue service interval.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This is the service interval used for comparison to generate Service Interval High and Service Interval OK events. See the *QServiceIntervalEvent* attribute.

Queue – QServiceInterval attribute

The value is in units of milliseconds, and is greater than or equal to zero, and less than or equal to 999 999 999.

To determine the value of this attribute, use the MQIA_Q_SERVICE_INTERVAL selector with the MQINQ call.

- This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value. This attribute is ignored on WebSphere Application Server embedded messaging using reduced function WebSphere MQ.

QServiceIntervalEvent (MQLONG)

Controls whether Service Interval High or Service Interval OK events are generated.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

- A Service Interval High event is generated when a check indicates that no messages have been retrieved from the queue for at least the time indicated by the *QServiceInterval* attribute.
- A Service Interval OK event is generated when a check indicates that messages have been retrieved from the queue within the time indicated by the *QServiceInterval* attribute.

Note: The value of this attribute can change dynamically.

The value is one of the following:

MQQSIE_HIGH

Queue Service Interval High events enabled.

- Queue Service Interval High events are **enabled** and
- Queue Service Interval OK events are **disabled**.

MQQSIE_OK

Queue Service Interval OK events enabled.

- Queue Service Interval High events are **disabled** and
- Queue Service Interval OK events are **enabled**.

MQQSIE_NONE

No queue service interval events enabled.

- Queue Service Interval High events are **disabled** and
- Queue Service Interval OK events are also **disabled**.

For shared queues, the value of this attribute is ignored; the value MQQSIE_NONE is assumed.

For more information about events, see the *WebSphere MQ Event Monitoring* book.

To determine the value of this attribute, use the MQIA_Q_SERVICE_INTERVAL_EVENT selector with the MQINQ call.

On z/OS, the MQINQ call cannot be used to determine the value of this attribute. This attribute is ignored on WebSphere Application Server embedded messaging using reduced function WebSphere MQ.

QSGDisp (MQLONG)

Queue-sharing group disposition.

Queue – QSGDisp attribute

Local	Model	Alias	Remote	Cluster
Yes	No	Yes	Yes	Yes

This specifies the disposition of the queue. The value is one of the following:

MQQSGD_Q_MGR

Queue manager disposition.

The object has queue-manager disposition. This means that the object definition is known only to the local queue manager; the definition is not known to other queue managers in the queue-sharing group.

It is possible for each queue manager in the queue-sharing group to have an object with the same name and type as the current object, but these are separate objects and there is no correlation between them. Their attributes are not constrained to be the same as each other.

MQQSGD_COPY

Copied-object disposition.

The object is a local copy of a master object definition that exists in the shared repository. Each queue manager in the queue-sharing group can have its own copy of the object. Initially, all copies have the same attributes, but by using MQSC commands each copy can be altered so that its attributes differ from those of the other copies. The attributes of the copies are resynchronized when the master definition in the shared repository is altered.

MQQSGD_SHARED

Shared disposition.

The object has shared disposition. This means that there exists in the shared repository a single instance of the object that is known to all queue managers in the queue-sharing group. When a queue manager in the group accesses the object, it accesses the single shared instance of the object.

To determine the value of this attribute, use the MQIA_QSG_DISP selector with the MQINQ call.

This attribute is supported only on z/OS.

QType (MQLONG)

Queue type.

Local	Model	Alias	Remote	Cluster
Yes	No	Yes	Yes	Yes

This attribute has one of the following values:

MQQT_ALIAS

Alias queue definition.

MQQT_CLUSTER

Cluster queue.

MQQT_LOCAL

Local queue.

MQQT_REMOTE

Local definition of a remote queue.

To determine the value of this attribute, use the MQIA_Q_TYPE selector with the MQINQ call.

RemoteQMgrName (MQCHAR48)

Name of remote queue manager.

Local	Model	Alias	Remote	Cluster
No	No	No	Yes	No

This is the name of the remote queue manager on which the queue *RemoteQName* is defined. If the *RemoteQName* queue has a *QSGDisp* value of MQQSGD_COPY or MQQSGD_SHARED, *RemoteQMgrName* can be the name of the queue-sharing group that owns *RemoteQName*.

If an application opens the local definition of a remote queue, *RemoteQMgrName* must not be blank and must not be the name of the local queue manager. If *XmitQName* is blank, the local queue whose name is the same as *RemoteQMgrName* is used as the transmission queue. If there is no queue with the name *RemoteQMgrName*, the queue identified by the *DefXmitQName* queue-manager attribute is used.

If this definition is used for a queue-manager alias, *RemoteQMgrName* is the name of the queue manager that is being aliased. It can be the name of the local queue manager. Otherwise, if *XmitQName* is blank when the open occurs, there must be a local queue whose name is the same as *RemoteQMgrName*; this queue is used as the transmission queue.

If this definition is used for a reply-to alias, this name is the name of the queue manager which is to be the *ReplyToQMgr*.

Note: No validation is performed on the value specified for this attribute when the queue definition is created or modified.

To determine the value of this attribute, use the MQCA_REMOTE_Q_MGR_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_MGR_NAME_LENGTH.

RemoteQName (MQCHAR48)

Name of remote queue.

Local	Model	Alias	Remote	Cluster
No	No	No	Yes	No

This is the name of the queue as it is known on the remote queue manager *RemoteQMgrName*.

If an application opens the local definition of a remote queue, when the open occurs *RemoteQName* must not be blank.

If this definition is used for a queue-manager alias definition, when the open occurs *RemoteQName* must be blank.

If the definition is used for a reply-to alias, this name is the name of the queue that is to be the *ReplyToQ*.

Queue – RemoteQName attribute

Note: No validation is performed on the value specified for this attribute when the queue definition is created or modified.

To determine the value of this attribute, use the MQCA_REMOTE_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

RetentionInterval (MQLONG)

Retention interval.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This is the period of time for which the queue should be retained. After this time has elapsed, the queue is eligible for deletion.

The time is measured in hours, counting from the date and time when the queue was created. The creation date and time of the queue are recorded in the *CreationDate* and *CreationTime* attributes, respectively.

This information is provided to enable a housekeeping application or the operator to identify and delete queues that are no longer required.

Note: The queue manager never takes any action to delete queues based on this attribute, or to prevent the deletion of queues whose retention interval has not expired; it is the user's responsibility to cause any required action to be taken.

A realistic retention interval should be used to prevent the accumulation of permanent dynamic queues (see *DefinitionType*). However, this attribute can also be used with predefined queues.

To determine the value of this attribute, use the MQIA_RETENTION_INTERVAL selector with the MQINQ call.

Scope (MQLONG)

Controls whether an entry for this queue also exists in a cell directory.

Local	Model	Alias	Remote	Cluster
Yes	No	Yes	Yes	No

A cell directory is provided by an installable Name service. The value is one of the following:

MQSCO_Q_MGR

Queue-manager scope.

The queue definition has queue-manager scope. This means that the definition of the queue does not extend beyond the queue manager which owns it. To open the queue for output from some other queue manager, either the name of the owning queue manager must be specified, or the other queue manager must have a local definition of the queue.

MQSCO_CELL

Cell scope.

Queue – Scope attribute

The queue definition has cell scope. This means that the queue definition is also placed in a cell directory available to all of the queue managers in the cell. The queue can be opened for output from any of the queue managers in the cell merely by specifying the name of the queue; the name of the queue manager which owns the queue need not be specified. However, the queue definition is not available to any queue manager in the cell which also has a local definition of a queue with that name, as the local definition takes precedence.

A cell directory is provided by an installable Name service. For example, the DCE Name service inserts the queue definition into the DCE directory.

Model and dynamic queues cannot have cell scope.

This value is only valid if a name service supporting a cell directory has been configured.

To determine the value of this attribute, use the MQIA_SCOPE selector with the MQINQ call.

Support for this attribute is subject to the following restrictions:

- On OS/400, the attribute is supported, but only MQSCO_Q_MGR is valid.
- On z/OS the attribute is not supported.

Shareability (MQLONG)

Whether queue can be shared for input.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This indicates whether the queue can be opened for input multiple times concurrently. The value is one of the following:

MQQA_SHAREABLE

Queue is shareable.

Multiple opens with the MQOO_INPUT_SHARED option are allowed.

MQQA_NOT_SHAREABLE

Queue is not shareable.

An MQOPEN call with the MQOO_INPUT_SHARED option is treated as MQOO_INPUT_EXCLUSIVE.

To determine the value of this attribute, use the MQIA_SHAREABILITY selector with the MQINQ call.

StorageClass (MQCHAR8)

Storage class for queue.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This is a user-defined name that defines the physical storage used to hold the queue. In practice, a message is written to disk only if it needs to be paged out of its memory buffer.

Queue – StorageClass attribute

To determine the value of this attribute, use the MQCA_STORAGE_CLASS selector with the MQINQ call. The length of this attribute is given by MQ_STORAGE_CLASS_LENGTH.

This attribute is supported only on z/OS.

TriggerControl (MQLONG)

Trigger control.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This controls whether trigger messages are written to an initiation queue, in order to cause an application to be started to service the queue. This is one of the following:

MQTC_OFF

Trigger messages not required.

No trigger messages are to be written for this queue. The value of *TriggerType* is irrelevant in this case.

MQTC_ON

Trigger messages required.

Trigger messages are to be written for this queue, when the appropriate trigger events occur.

To determine the value of this attribute, use the MQIA_TRIGGER_CONTROL selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

TriggerData (MQCHAR64)

Trigger data.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This is free-format data that the queue manager inserts into the trigger message when a message arriving on this queue causes a trigger message to be written to the initiation queue.

The content of this data is of no significance to the queue manager. It is meaningful either to the trigger-monitor application which processes the initiation queue, or to the application which is started by the trigger monitor.

The character string cannot contain any nulls. It is padded to the right with blanks if necessary.

To determine the value of this attribute, use the MQCA_TRIGGER_DATA selector with the MQINQ call. To change the value of this attribute, use the MQSET call. The length of this attribute is given by MQ_TRIGGER_DATA_LENGTH.

This attribute is not supported in the following environments: Windows 3.1, Windows 95, Windows 98.

TriggerDepth (MQLONG)

Trigger depth.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This is the number of messages of priority *TriggerMsgPriority* or greater that must be on the queue before a trigger message is written. This applies when *TriggerType* is set to MQTT_DEPTH. The value of *TriggerDepth* is one or greater. This attribute is not used otherwise.

To determine the value of this attribute, use the MQIA_TRIGGER_DEPTH selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

TriggerMsgPriority (MQLONG)

Threshold message priority for triggers.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This is the message priority below which messages do not contribute to the generation of trigger messages (that is, the queue manager ignores these messages when deciding whether a trigger message should be generated). *TriggerMsgPriority* can be in the range zero (lowest) through *MaxPriority* (highest; see Chapter 43, “Attributes for the queue manager”, on page 501); a value of zero causes all messages to contribute to the generation of trigger messages.

To determine the value of this attribute, use the MQIA_TRIGGER_MSG_PRIORITY selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

TriggerType (MQLONG)

Trigger type.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This controls the conditions under which trigger messages are written as a result of messages arriving on this queue. The value is one of the following:

MQTT_NONE

No trigger messages.

No trigger messages are written as a result of messages on this queue. This has the same effect as setting *TriggerControl* to MQTC_OFF.

MQTT_FIRST

Trigger message when queue depth goes from 0 to 1.

A trigger message is written whenever the number of messages of priority *TriggerMsgPriority* or greater on the queue changes from 0 to 1.

MQTT EVERY

Trigger message for every message.

Queue – TriggerType attribute

A trigger message is written whenever a message of priority *TriggerMsgPriority* or greater arrives on the queue.

MQTT_DEPTH

Trigger message when depth threshold exceeded.

A trigger message is written whenever the number of messages of priority *TriggerMsgPriority* or greater on the queue equals or exceeds *TriggerDepth*. After the trigger message has been written, *TriggerControl* is set to *MQTC_OFF* to prevent further triggering until it is explicitly turned on again.

To determine the value of this attribute, use the *MQIA_TRIGGER_TYPE* selector with the *MQINQ* call. To change the value of this attribute, use the *MQSET* call.

Usage (MQLONG)

Queue usage.

Local	Model	Alias	Remote	Cluster
Yes	Yes	No	No	No

This indicates what the queue is used for. The value is one of the following:

MQUS_NORMAL

Normal usage.

This is a queue that normal applications use when putting and getting messages; the queue is not a transmission queue.

MQUS_TRANSMISSION

Transmission queue.

This is a queue used to hold messages destined for remote queue managers. When a normal application sends a message to a remote queue, the local queue manager stores the message temporarily on the appropriate transmission queue in a special format. A message channel agent then reads the message from the transmission queue, and transports the message to the remote queue manager. For more information about transmission queues, see the *WebSphere MQ Application Programming Guide*.

Only privileged applications can open a transmission queue for *MQOO_OUTPUT* to put messages on it directly. Only utility applications would normally be expected to do this. Care must be taken that the message data format is correct (see Chapter 25, “MQXQH – Transmission-queue header”, on page 317), otherwise errors may occur during the transmission process. Context is not passed or set unless one of the *MQPMO_*_CONTEXT* context options is specified.

To determine the value of this attribute, use the *MQIA_USAGE* selector with the *MQINQ* call.

XmitQName (MQCHAR48)

Transmission queue name.

Local	Model	Alias	Remote	Cluster
No	No	No	Yes	No

Queue – XmitQName attribute

If this attribute is nonblank when an open occurs, either for a remote queue or for a queue-manager alias definition, it specifies the name of the local transmission queue to be used for forwarding the message.

If *XmitQName* is blank, the local queue whose name is the same as *RemoteQMGrName* is used as the transmission queue. If there is no queue with the name *RemoteQMGrName*, the queue identified by the *DefXmitQName* queue-manager attribute is used.

This attribute is ignored if the definition is being used as a queue-manager alias and *RemoteQMGrName* is the name of the local queue manager. It is also ignored if the definition is used as a reply-to queue alias definition.

To determine the value of this attribute, use the MQCA_XMIT_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

Object attributes

Chapter 41. Attributes for namelists

Namelists are supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq Tru64 UNIX, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

The following table summarizes the attributes that are specific to namelists. The attributes are described in alphabetic order.

Note: The names of the attributes shown in this book are the names used with the MQINQ and MQSET calls. When MQSC commands are used to define, alter, or display attributes, alternative short names are used; see the *WebSphere MQ Script (MQSC) Command Reference* for details.

Table 95. Attributes for namelists

Attribute	Description	Page
<i>AlterationDate</i>	Date when definition was last changed	491
<i>AlterationTime</i>	Time when definition was last changed	491
<i>NameCount</i>	Number of names in namelist	492
<i>NamelistDesc</i>	Namelist description	492
<i>NamelistName</i>	Namelist name	492
<i>Names</i>	A list of <i>NameCount</i> names	493
<i>NamelistType</i>	Namelist type	493
<i>QSGDisp</i>	Queue-sharing group disposition	493

Attribute descriptions

A namelist object has the attributes described below.

AlterationDate (MQCHAR12)

Date when definition was last changed.

This is the date when the definition was last changed. The format of the date is YYYY-MM-DD, padded with two trailing blanks to make the length 12 bytes.

To determine the value of this attribute, use the MQCA_ALTERATION_DATE selector with the MQINQ call. The length of this attribute is given by MQ_DATE_LENGTH.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems. In MQSeries for Compaq NonStop Kernel, V5.1, the AlterationDate attribute is updated only when administrative changes are made to attributes of the namelist.

AlterationTime (MQCHAR8)

Time when definition was last changed.

Namelist – AlterationTime attribute

This is the time when the definition was last changed. The format of the time is HH.MM.SS.

To determine the value of this attribute, use the MQCA_ALTERATION_TIME selector with the MQINQ call. The length of this attribute is given by MQ_TIME_LENGTH.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems. In MQSeries for Compaq NonStop Kernel, V5.1, the AlterationTime attribute is updated only when administrative changes are made to attributes of the namelist.

NameCount (MQLONG)

Number of names in namelist.

This is greater than or equal to zero. The following value is defined:

MQNC_MAX_NAMELIST_NAME_COUNT

Maximum number of names in a namelist.

To determine the value of this attribute, use the MQIA_NAME_COUNT selector with the MQINQ call.

NamelistDesc (MQCHAR64)

Namelist description.

This is a field that may be used for descriptive commentary; its value is established by the definition process. The content of the field is of no significance to the queue manager, but the queue manager may require that the field contain only characters that can be displayed. It cannot contain any null characters; if necessary, it is padded to the right with blanks. In a DBCS installation, this field can contain DBCS characters (subject to a maximum field length of 64 bytes).

Note: If this field contains characters that are not in the queue manager's character set (as defined by the *CodedCharSetId* queue manager attribute), those characters may be translated incorrectly if this field is sent to another queue manager.

To determine the value of this attribute, use the MQCA_NAMELIST_DESC selector with the MQINQ call.

The length of this attribute is given by MQ_NAMELIST_DESC_LENGTH.

NamelistName (MQCHAR48)

Namelist name.

This is the name of a namelist that is defined on the local queue manager. For more information about namelist names, see the *WebSphere MQ Application Programming Guide*.

Each namelist has a name that is different from the names of other namelists belonging to the queue manager, but may duplicate the names of other queue manager objects of different types (for example, queues).

Namelist – NamelistName attribute

To determine the value of this attribute, use the MQCA_NAMELIST_NAME selector with the MQINQ call.

The length of this attribute is given by MQ_NAMELIST_NAME_LENGTH.

NamelistType (MQLONG)

Namelist type.

This specifies the nature of the names in the namelist, and indicates how the namelist is used. The value is one of the following:

MQNT_NONE

Namelist with no assigned type.

MQNT_Q

Namelist containing the names of queues.

MQNT_CLUSTER

Namelist containing the names of clusters.

MQNT_AUTH_INFO

Namelist containing the names of authentication-information objects.

To determine the value of this attribute, use the MQIA_NAMELIST_TYPE selector with the MQINQ call.

This attribute is supported only on z/OS.

Names (MQCHAR48xNameCount)

A list of *NameCount* names.

Each name is the name of an object that is defined to the local queue manager. For more information about object names, see the *WebSphere MQ Application Programming Guide*.

To determine the value of this attribute, use the MQCA_NAMES selector with the MQINQ call.

The length of each name in the list is given by MQ_OBJECT_NAME_LENGTH.

QSGDisp (MQLONG)

Queue-sharing group disposition.

This specifies the disposition of the namelist. The value is one of the following:

MQQSGD_Q_MGR

Queue manager disposition.

The object has queue-manager disposition. This means that the object definition is known only to the local queue manager; the definition is not known to other queue managers in the queue-sharing group.

It is possible for each queue manager in the queue-sharing group to have an object with the same name and type as the current object, but these are separate objects and there is no correlation between them. Their attributes are not constrained to be the same as each other.

Namelist – QSGDisp attribute

MQQSGD_COPY

Copied-object disposition.

The object is a local copy of a master object definition that exists in the shared repository. Each queue manager in the queue-sharing group can have its own copy of the object. Initially, all copies have the same attributes, but by using MQSC commands each copy can be altered so that its attributes differ from those of the other copies. The attributes of the copies are resynchronized when the master definition in the shared repository is altered.

To determine the value of this attribute, use the MQIA_QSG_DISP selector with the MQINQ call.

This attribute is supported only on z/OS.

Chapter 42. Attributes for process definitions

Process definitions are not supported in the following environments: VSE/ESA, Windows 3.1, Windows 95, Windows 98.

The following table summarizes the attributes that are specific to process definitions. The attributes are described in alphabetic order.

Note: The names of the attributes shown in this book are the names used with the MQINQ and MQSET calls. When MQSC commands are used to define, alter, or display attributes, alternative short names are used; see the *WebSphere MQ Script (MQSC) Command Reference* for details.

Table 96. Attributes for process definitions

Attribute	Description	Page
<i>AlterationDate</i>	Date when definition was last changed	495
<i>AlterationTime</i>	Time when definition was last changed	495
<i>AppId</i>	Application identifier	496
<i>AppType</i>	Application type	496
<i>EnvData</i>	Environment data	497
<i>ProcessDesc</i>	Process description	498
<i>ProcessName</i>	Process name	498
<i>QSGDisp</i>	Queue-sharing group disposition	498
<i>UserData</i>	User data	499

Attribute descriptions

A process-definition object has the attributes described below.

AlterationDate (MQCHAR12)

Date when definition was last changed.

This is the date when the definition was last changed. The format of the date is YYYY-MM-DD, padded with two trailing blanks to make the length 12 bytes.

To determine the value of this attribute, use the MQCA_ALTERATION_DATE selector with the MQINQ call. The length of this attribute is given by MQ_DATE_LENGTH.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems. In MQSeries for Compaq NonStop Kernel, V5.1, the AlterationDate attribute is updated only when administrative changes are made to attributes of the process.

AlterationTime (MQCHAR8)

Time when definition was last changed.

Process definition – AlterationTime attribute

This is the time when the definition was last changed. The format of the time is HH.MM.SS.

To determine the value of this attribute, use the MQCA_ALTERATION_TIME selector with the MQINQ call. The length of this attribute is given by MQ_TIME_LENGTH.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems. In MQSeries for Compaq NonStop Kernel, V5.1, the AlterationTime attribute is updated only when administrative changes are made to attributes of the process.

ApplId (MQCHAR256)

Application identifier.

This is a character string that identifies the application to be started. This information is for use by a trigger-monitor application that processes messages on the initiation queue; the information is sent to the initiation queue as part of the trigger message.

The meaning of *ApplId* is determined by the trigger-monitor application. The trigger monitor provided by MQSeries requires *ApplId* to be the name of an executable program. The following notes apply to the environments indicated:

- On z/OS, *ApplId* must be:
 - A CICS transaction identifier, for applications started using the CICS trigger-monitor transaction CKTI
 - An IMS transaction identifier, for applications started using the IMS trigger monitor CSQQTRMN
- On PC DOS, OS/2, and Windows systems, the program name can be prefixed with a drive and directory path.
- On Compaq OpenVMS Alpha, the program name can be prefixed with a directory path.
- On Compaq NonStop Kernel, the program name can be prefixed with a volume/subvolume location.
- On UNIX systems, the program name can be prefixed with a directory path.

The character string cannot contain any nulls. It is padded to the right with blanks if necessary.

To determine the value of this attribute, use the MQCA_APPL_ID selector with the MQINQ call. The length of this attribute is given by MQ_PROCESS_APPL_ID_LENGTH.

AppType (MQLONG)

Application type.

This identifies the nature of the program to be started in response to the receipt of a trigger message. This information is for use by a trigger-monitor application that processes messages on the initiation queue; the information is sent to the initiation queue as part of the trigger message.

Process definition – ApplType attribute

ApplType can have any value, but the following values are recommended for standard types; user-defined application types should be restricted to values in the range MQAT_USER_FIRST through MQAT_USER_LAST:

MQAT_AIX

AIX application (same value as MQAT_UNIX).

MQAT_CICS

CICS transaction.

MQAT_DOS

WebSphere MQ client application on PC DOS.

MQAT_IMS

IMS application.

MQAT_MVS

MVS or TSO application (same value as MQAT_ZOS).

MQAT_NOTES_AGENT

Lotus Notes Agent application.

MQAT_NSK

Compaq NonStop Kernel application.

MQAT_OS2

OS/2 or Presentation Manager application.

MQAT_OS390

OS/390 application (same value as MQAT_ZOS).

MQAT_OS400

OS/400 application.

MQAT_UNIX

UNIX application.

MQAT_VMS

Digital OpenVMS application.

MQAT_WINDOWS

16-bit Windows application.

MQAT_WINDOWS_NT

32-bit Windows application.

MQAT_WLM

z/OS workload manager application.

MQAT_ZOS

z/OS application.

MQAT_USER_FIRST

Lowest value for user-defined application type.

MQAT_USER_LAST

Highest value for user-defined application type.

To determine the value of this attribute, use the MQIA_APPL_TYPE selector with the MQINQ call.

EnvData (MQCHAR128)

Environment data.

This is a character string that contains environment-related information pertaining to the application to be started. This information is for use by a trigger-monitor application that processes messages on the initiation queue; the information is sent to the initiation queue as part of the trigger message.

The meaning of *EnvData* is determined by the trigger-monitor application. The trigger monitor provided by MQSeries appends *EnvData* to the parameter list passed to the started application. The parameter list consists of the MQTMC2

Process definition – EnvData attribute

structure, followed by one blank, followed by *EnvData* with trailing blanks removed. The following notes apply to the environments indicated:

- On z/OS, *EnvData* is not used by the trigger-monitor applications provided by WebSphere MQ.
- On z/OS, if *ApplType* is MQAT_WLM, you can supply default values in *EnvData* for the *ServiceName* and *ServiceStep* fields in the work information header (MQWIH).
- On UNIX systems, *EnvData* can be set to the & character to cause the started application to run in the background.

The character string cannot contain any nulls. It is padded to the right with blanks if necessary.

To determine the value of this attribute, use the MQCA_ENV_DATA selector with the MQINQ call. The length of this attribute is given by MQ_PROCESS_ENV_DATA_LENGTH.

ProcessDesc (MQCHAR64)

Process description.

This is a field that may be used for descriptive commentary. The content of the field is of no significance to the queue manager, but the queue manager may require that the field contain only characters that can be displayed. It cannot contain any null characters; if necessary, it is padded to the right with blanks. In a DBCS installation, the field can contain DBCS characters (subject to a maximum field length of 64 bytes).

Note: If this field contains characters that are not in the queue manager's character set (as defined by the *CodedCharSetId* queue manager attribute), those characters may be translated incorrectly if this field is sent to another queue manager.

To determine the value of this attribute, use the MQCA_PROCESS_DESC selector with the MQINQ call.

The length of this attribute is given by MQ_PROCESS_DESC_LENGTH.

ProcessName (MQCHAR48)

Process name.

This is the name of a process definition that is defined on the local queue manager.

Each process definition has a name that is different from the names of other process definitions belonging to the queue manager. But the name of the process definition may be the same as the names of other queue manager objects of different types (for example, queues).

To determine the value of this attribute, use the MQCA_PROCESS_NAME selector with the MQINQ call.

The length of this attribute is given by MQ_PROCESS_NAME_LENGTH.

QSGDisp (MQLONG)

Queue-sharing group disposition.

Process definition – QSGDisp attribute

This specifies the disposition of the process definition. The value is one of the following:

MQQSGD_Q_MGR

Queue manager disposition.

The object has queue-manager disposition. This means that the object definition is known only to the local queue manager; the definition is not known to other queue managers in the queue-sharing group.

It is possible for each queue manager in the queue-sharing group to have an object with the same name and type as the current object, but these are separate objects and there is no correlation between them. Their attributes are not constrained to be the same as each other.

MQQSGD_COPY

Copied-object disposition.

The object is a local copy of a master object definition that exists in the shared repository. Each queue manager in the queue-sharing group can have its own copy of the object. Initially, all copies have the same attributes, but by using MQSC commands each copy can be altered so that its attributes differ from those of the other copies. The attributes of the copies are resynchronized when the master definition in the shared repository is altered.

To determine the value of this attribute, use the MQIA_QSG_DISP selector with the MQINQ call.

This attribute is supported only on z/OS.

UserData (MQCHAR128)

User data.

This is a character string that contains user information pertaining to the application to be started. This information is for use by a trigger-monitor application that processes messages on the initiation queue, or the application which is started by the trigger monitor. The information is sent to the initiation queue as part of the trigger message.

The meaning of *UserData* is determined by the trigger-monitor application. The trigger monitor provided by MQSeries simply passes *UserData* to the started application as part of the parameter list. The parameter list consists of the MQTMC2 structure (containing *UserData*), followed by one blank, followed by *EnvData* with trailing blanks removed.

The character string cannot contain any nulls. It is padded to the right with blanks if necessary.

To determine the value of this attribute, use the MQCA_USER_DATA selector with the MQINQ call. The length of this attribute is given by MQ_PROCESS_USER_DATA_LENGTH.

Object attributes

Chapter 43. Attributes for the queue manager

Queue-manager attributes are not supported on VSE/ESA.

Some queue-manager attributes are fixed for particular implementations, while others can be changed by using the MQSC command ALTER QMGR. The attributes can also be displayed by using the command DISPLAY QMGR. Most queue-manager attributes can be inquired by opening a special MQOT_Q_MGR object, and using the MQINQ call with the handle returned.

The following table summarizes the attributes that are specific to the queue manager. The attributes are described in alphabetic order.

Note: The names of the attributes shown in this book are the names used with the MQINQ and MQSET calls. When MQSC commands are used to define, alter, or display attributes, alternative short names are used; see the *WebSphere MQ Script (MQSC) Command Reference* for details.

Table 97. Attributes for the queue manager

Attribute	Description	Page
<i>AlterationDate</i>	Date when definition was last changed	502
<i>AlterationTime</i>	Time when definition was last changed	503
<i>AuthorityEvent</i>	Controls whether authorization (Not Authorized) events are generated	503
<i>ChannelAutoDef</i>	Controls whether automatic channel definition is permitted	503
<i>ChannelAutoDefEvent</i>	Controls whether channel automatic-definition events are generated	504
<i>ChannelAutoDefExit</i>	Name of user exit for automatic channel definition	504
<i>ClusterWorkloadData</i>	User data for cluster workload exit	505
<i>ClusterWorkloadExit</i>	Name of user exit for cluster workload management	505
<i>ClusterWorkloadLength</i>	Maximum length of message data passed to cluster workload exit	505
<i>CodedCharSetId</i>	Coded character set identifier	506
<i>CommandInputQName</i>	Command input queue name	506
<i>CommandLevel</i>	Command level	506
<i>DeadLetterQName</i>	Name of dead-letter queue	508
<i>DefXmitQName</i>	Default transmission queue name	510
<i>DistLists</i>	Distribution list support	510
<i>ExpiryInterval</i>	Interval between scans for expired messages	510
<i>IGQPutAuthority</i>	Intra-group queuing put authority	510
<i>IGQUserId</i>	Intra-group queuing user identifier	512
<i>InhibitEvent</i>	Controls whether inhibit (Inhibit Get and Inhibit Put) events are generated	512
<i>IntraGroupQueuing</i>	Intra-group queuing support	512

Attributes – queue manager

Table 97. Attributes for the queue manager (continued)

Attribute	Description	Page
<i>LocalEvent</i>	Controls whether local error events are generated	513
<i>MaxHandles</i>	Maximum number of handles	513
<i>MaxMsgLength</i>	Maximum message length in bytes	514
<i>MaxPriority</i>	Maximum priority	514
<i>MaxUncommittedMsgs</i>	Maximum number of uncommitted messages within a unit of work	514
<i>PerformanceEvent</i>	Controls whether performance-related events are generated	515
<i>Platform</i>	Platform on which the queue manager is running	516
<i>QMgrDesc</i>	Queue manager description	516
<i>QMgrIdentifier</i>	Unique internally-generated identifier of queue manager	517
<i>QMgrName</i>	Queue manager name	517
<i>QSGName</i>	Name of queue-sharing group	517
<i>RemoteEvent</i>	Controls whether remote error events are generated	517
<i>RepositoryName</i>	Name of cluster for which this queue manager provides repository services	518
<i>RepositoryNamelist</i>	Name of namelist object containing names of clusters for which this queue manager provides repository services	518
<i>SSLCRLNamelist</i>	Name of namelist object containing names of authentication information objects.	Note 1
<i>SSLCryptoHardware</i>	Cryptographic hardware configuration string.	Note 1
<i>SSLKeyRepository</i>	Location of SSL key repository.	Note 1
<i>SSLTasks</i>	Number of server subtasks for processing SSL calls.	Note 1
<i>StartStopEvent</i>	Controls whether start and stop events are generated	519
<i>SyncPoint</i>	Syncpoint availability	519
<i>TriggerInterval</i>	Trigger-message interval	519
Notes:		
1. This attribute cannot be inquired using the MQINQ call, and is not described in this book. See the <i>WebSphere MQ Programmable Command Formats and Administration Interface</i> book for details of this attribute.		

Attribute descriptions

The queue-manager object has the attributes described below.

AlterationDate (MQCHAR12)

Date when definition was last changed.

This is the date when the definition was last changed. The format of the date is YYYY-MM-DD, padded with two trailing blanks to make the length 12 bytes.

Queue manager – AlterationDate attribute

To determine the value of this attribute, use the MQCA_ALTERATION_DATE selector with the MQINQ call. The length of this attribute is given by MQ_DATE_LENGTH.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems. In MQSeries for Compaq NonStop Kernel, V5.1, the AlterationDate attribute is updated only when administrative changes are made to attributes of the queue manager.

AlterationTime (MQCHAR8)

Time when definition was last changed.

This is the time when the definition was last changed. The format of the time is HH.MM.SS.

To determine the value of this attribute, use the MQCA_ALTERATION_TIME selector with the MQINQ call. The length of this attribute is given by MQ_TIME_LENGTH.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems. In MQSeries for Compaq NonStop Kernel, V5.1, the AlterationTime attribute is updated only when administrative changes are made to attributes of the queue manager.

AuthorityEvent (MQLONG)

Controls whether authorization (Not Authorized) events are generated.

The value is one of the following:

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

For more information about events, see the *WebSphere MQ Event Monitoring* book.

To determine the value of this attribute, use the MQIA_AUTHORITY_EVENT selector with the MQINQ call.

- On z/OS, the MQINQ call cannot be used to determine the value of this attribute, and the attribute is always in the disabled state.

ChannelAutoDef (MQLONG)

Controls whether automatic channel definition is permitted.

This attribute controls the automatic definition of channels of type MQCHT_RECEIVER and MQCHT_SVRCONN. Note that the automatic definition of MQCHT_CLUSSDR channels is always enabled. (On WebSphere Application Server embedded messaging using reduced function WebSphere MQ, only SVRCONN channels are available.) The value is one of the following:

MQCHAD_DISABLED

Channel auto-definition disabled.

Queue manager – ChannelAutoDef attribute

MQCHAD_ENABLED

Channel auto-definition enabled.

To determine the value of this attribute, use the MQIA_CHANNEL_AUTO_DEF selector with the MQINQ call.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

ChannelAutoDefEvent (MQLONG)

Controls whether channel automatic-definition events are generated.

This applies to channels of type MQCHT_RECEIVER, MQCHT_SVRCONN, and MQCHT_CLUSSDR. The value is one of the following:

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

For more information about events, see the *WebSphere MQ Event Monitoring* book.

To determine the value of this attribute, use the MQIA_CHANNEL_AUTO_DEF_EVENT selector with the MQINQ call.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

- On z/OS, the MQINQ call cannot be used to determine the value of this attribute. You cannot enable event attributes on WebSphere Application Server embedded messaging using reduced function WebSphere MQ.

ChannelAutoDefExit (MQCHARn)

Name of user exit for automatic channel definition.

If this name is nonblank, and *ChannelAutoDef* has the value MQCHAD_ENABLED, the exit is called each time that the queue manager is about to create a channel definition. This applies to channels of type MQCHT_RECEIVER, MQCHT_SVRCONN, and MQCHT_CLUSSDR. The exit can then do one of the following:

- Allow the creation of the channel definition to proceed without change.
- Modify the attributes of the channel definition that is created.
- Suppress creation of the channel entirely.

Note: Both the length and the value of this attribute are environment specific. See the introduction to the MQCD structure in the *WebSphere MQ Intercommunication* book for details of the value of this attribute in various environments.

To determine the value of this attribute, use the MQCA_CHANNEL_AUTO_DEF_EXIT selector with the MQINQ call. The length of this attribute is given by MQ_EXIT_NAME_LENGTH.

Queue manager – ChannelAutoDefExit attribute

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

ClusterWorkloadData (MQCHAR32)

User data for cluster workload exit.

This is a user-defined 32-byte character string that is passed to the cluster workload exit when it is called. If there is no data to pass to the exit, the string is blank.

To determine the value of this attribute, use the MQCA_CLUSTER_WORKLOAD_DATA selector with the MQINQ call.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

ClusterWorkloadExit (MQCHARn)

Name of user exit for cluster workload management.

If this name is nonblank, the exit is called each time that a message is put to a cluster queue or moved from one cluster-sender queue to another. The exit can then decide whether to accept the queue instance selected by the queue manager as the destination for the message, or choose another queue instance.

Note: Both the length and the value of this attribute are environment specific. See the *WebSphere MQ Intercommunication* manual for details of the value of this attribute in various environments.

To determine the value of this attribute, use the MQCA_CLUSTER_WORKLOAD_EXIT selector with the MQINQ call. The length of this attribute is given by MQ_EXIT_NAME_LENGTH.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

ClusterWorkloadLength (MQLONG)

Maximum length of message data passed to cluster workload exit.

This is the maximum length of message data that is passed to the cluster workload exit. The actual length of data passed to the exit is the minimum of the following:

- The length of the message.
- The queue-manager's *MaxMsgLength* attribute.
- The *ClusterWorkloadLength* attribute.

To determine the value of this attribute, use the MQIA_CLUSTER_WORKLOAD_LENGTH selector with the MQINQ call.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Queue manager – CodedCharSetId attribute

CodedCharSetId (MQLONG)

Coded character set identifier.

This defines the character set used by the queue manager for all character string fields defined in the MQI, including the names of objects, queue creation date and time, and so on. The character set must be one that has single-byte characters for the characters that are valid in object names. It does not apply to application data carried in the message. The value depends on the environment:

- On z/OS, the value is set from the system parameters when the queue manager is started; the default value is 500. Refer to the *WebSphere MQ for z/OS System Setup Guide* for further information.
- On OS/2 and Windows, the value is the primary CODEPAGE of the user creating the queue manager.
- On OS/400, the value is that which is set in the environment when the queue manager is first created.
- On Compaq OpenVMS Alpha, Compaq NonStop Kernel, and UNIX systems, the value is the default CODESET for the locale of the user creating the queue manager.

To determine the value of this attribute, use the MQIA_CODED_CHAR_SET_ID selector with the MQINQ call.

CommandInputQName (MQCHAR48)

Command input queue name.

This is the name of the command input queue defined on the local queue manager. This is a queue to which users can send commands, if authorized to do so. The name of the queue depends on the environment:

- On z/OS, the name of the queue is SYSTEM.COMMAND.INPUT, and only MQSC commands can be sent to it. Refer to the *WebSphere MQ Script (MQSC) Command Reference* book for details of MQSC commands.
- In all other environments, the name of the queue is SYSTEM.ADMIN.COMMAND.QUEUE, and only PCF commands can be sent to it. However, an MQSC command can be sent to this queue if the MQSC command is enclosed within a PCF command of type MQCMD_ESCAPE. Refer to the *WebSphere MQ Programmable Command Formats and Administration Interface* book for details of the Escape command.

To determine the value of this attribute, use the MQCA_COMMAND_INPUT_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

CommandLevel (MQLONG)

Command Level.

This indicates the level of system control commands supported by the queue manager. The value is one of the following:

MQCMDL_LEVEL_1

Level 1 of system control commands.

This value is returned by the following:

- MQSeries for AIX Version 2 Release 2
- MQSeries for MVS/ESA™

Queue manager – CommandLevel attribute

- Version 1 Release 1.1
- Version 1 Release 1.2
- Version 1 Release 1.3
- MQSeries for OS/2 Version 2 Release 0
- MQSeries for OS/400
 - Version 2 Release 3
 - Version 3 Release 1
 - Version 3 Release 6
- MQSeries for Windows Version 2 Release 0

MQCMDL_LEVEL_101

MQSeries for Windows Version 2 Release 0.1.

MQCMDL_LEVEL_110

MQSeries for Windows Version 2 Release 1.

MQCMDL_LEVEL_114

MQSeries for MVS/ESA Version 1 Release 1.4.

MQCMDL_LEVEL_120

MQSeries for MVS/ESA Version 1 Release 2.0.

MQCMDL_LEVEL_200

MQSeries for Windows NT Version 2 Release 0.

MQCMDL_LEVEL_201

MQSeries for OS/2 Version 2 Release 0.1.

MQCMDL_LEVEL_210

MQSeries for OS/390 Version 2 Release 1.0.

MQCMDL_LEVEL_220

Level 220 of system control commands.

This value is returned by the following:

- MQSeries for AT&T GIS UNIX Version 2 Release 2
- MQSeries for SINIX and DC/OSx Version 2 Release 2
- MQSeries for SunOS Version 2 Release 2
- MQSeries for Tandem NonStop Kernel Version 2 Release 2

MQCMDL_LEVEL_221

Level 221 of system control commands.

This value is returned by the following:

- MQSeries for AIX Version 2 Release 2.1
- MQSeries for Digital OpenVMS Version 2 Release 2.1

MQCMDL_LEVEL_320

Level 320 of system control commands.

This value is returned by the following:

- MQSeries for OS/400
 - Version 3 Release 2
 - Version 3 Release 7

MQCMDL_LEVEL_420

Level 420 of system control commands.

This value is returned by the following:

- MQSeries for AS/400
 - Version 4 Release 2.0
 - Version 4 Release 2.1

Queue manager – CommandLevel attribute

MQCMDL_LEVEL_500

Level 500 of system control commands.

This value is returned by the following:

- MQSeries for AIX Version 5 Release 0
- MQSeries for HP-UX Version 5 Release 0
- MQSeries for OS/2 Version 5 Release 0
- MQSeries for Solaris Version 5 Release 0
- MQSeries for Windows NT Version 5 Release 0

MQCMDL_LEVEL_510

Level 510 of system control commands.

This value is returned by the following:

- MQSeries for AIX Version 5 Release 1
- MQSeries for AS/400 Version 5 Release 1
- MQSeries for HP-UX Version 5 Release 1
- MQSeries for OS/2 Version 5 Release 1
- MQSeries for Compaq OpenVMS Alpha Version 5 Release 1
- MQSeries for Compaq NonStop Kernel Version 5 Release 1
- MQSeries for Compaq Tru64 UNIX Version 5 Release 1
- MQSeries for Solaris Version 5 Release 1
- MQSeries for Windows NT Version 5 Release 1

MQCMDL_LEVEL_520

Level 520 of system control commands.

This value is returned by the following:

- MQSeries for AIX Version 5 Release 2
- MQSeries for AS/400 Version 5 Release 2
- MQSeries for HP-UX Version 5 Release 2
- MQSeries for Linux Version 5 Release 2
- MQSeries for OS/390 Version 5 Release 2
- MQSeries for Sun Solaris Version 5 Release 2
- MQSeries for Windows NT Version 5 Release 2

MQCMDL_LEVEL_530

Level 530 of system control commands.

This value is returned by the following:

- Websphere MQ for AIX Version 5 Release 3
- Websphere MQ for HP-UX Version 5 Release 3
- Websphere MQ for i-Series Version 5 Release 3
- WebSphere MQ for Linux for Intel Version 5 Release 3
- WebSphere MQ for Linux for zSeries Version 5 Release 3
- Websphere MQ for Sun Solaris Version 5 Release 3
- Websphere MQ for Windows Version 5 Release 3
- Websphere MQ for z/OS Version 5 Release 3

The set of system control commands that corresponds to a particular value of the *CommandLevel* attribute varies according to the value of the *Platform* attribute; both must be used to decide which system control commands are supported.

To determine the value of this attribute, use the MQIA_COMMAND_LEVEL selector with the MQINQ call.

DeadLetterQName (MQCHAR48)

Name of dead-letter (undelivered-message) queue.

Queue manager – DeadLetterQName attribute

This is the name of a queue defined on the local queue manager. Messages are sent to this queue if they cannot be routed to their correct destination.

For example, messages are put on this queue when:

- A message arrives at a queue manager, destined for a queue that is not yet defined on that queue manager
- A message arrives at a queue manager, but the queue for which it is destined cannot receive it because, possibly:
 - The queue is full
 - Put requests are inhibited
 - The sending node does not have authority to put messages on the queue

Applications can also put messages on the dead-letter queue.

Report messages are treated in the same way as ordinary messages; if the report message cannot be delivered to its destination queue (usually the queue specified by the *ReplyToQ* field in the message descriptor of the original message), the report message is placed on the dead-letter (undelivered-message) queue.

Note: Messages that have passed their expiry time (see the *Expiry* field described in Chapter 10, “MQMD – Message descriptor”, on page 141) are **not** transferred to this queue when they are discarded. However, an expiration report message (MQRO_EXPIRATION) is still generated and sent to the *ReplyToQ* queue, if requested by the sending application.

Messages are not put on the dead-letter (undelivered-message) queue when the application that issued the put request has been notified synchronously of the problem by means of the reason code returned by the MQPUT or MQPUT1 call (for example, a message put on a local queue for which put requests are inhibited).

Messages on the dead-letter (undelivered-message) queue sometimes have their application message data prefixed with an MQDLH structure. This structure contains extra information that indicates why the message was placed on the dead-letter (undelivered-message) queue. See Chapter 7, “MQDLH – Dead-letter header”, on page 83 for more details of this structure.

This queue must be a local queue, with a *Usage* attribute of MQUS_NORMAL.

If a dead-letter (undelivered-message) queue is not supported by a queue manager, or one has not been defined, the name is all blanks. All WebSphere MQ queue managers support a dead-letter (undelivered-message) queue, but by default it is not defined.

If the dead-letter (undelivered-message) queue is not defined, or it is full, or unusable for some other reason, a message which would have been transferred to it by a message channel agent is retained instead on the transmission queue.

To determine the value of this attribute, use the MQCA_DEAD_LETTER_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

This attribute is not supported in the following environments: Windows 3.1, Windows 95, Windows 98.

Queue manager – DefXmitQName attribute

DefXmitQName (MQCHAR48)

Default transmission queue name.

This is the name of the transmission queue that is used for the transmission of messages to remote queue managers, if there is no other indication of which transmission queue to use.

If there is no default transmission queue, the name is entirely blank. The initial value of this attribute is blank.

To determine the value of this attribute, use the MQCA_DEF_XMIT_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

DistLists (MQLONG)

Distribution list support.

This indicates whether the local queue manager supports distribution lists on the MQPUT and MQPUT1 calls. The value is one of the following:

MQDL_SUPPORTED

Distribution lists supported.

MQDL_NOT_SUPPORTED

Distribution lists not supported.

To determine the value of this attribute, use the MQIA_DIST_LISTS selector with the MQINQ call.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, Compaq Tru64 UNIX, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

ExpiryInterval (MQLONG)

Interval between scans for expired messages.

This indicates the frequency with which the queue manager scans the queues looking for expired messages. It is either a time interval in seconds in the range 1 through 99 999 999, or the following special value:

MQEXPI_OFF

No scans for expired messages.

This indicates that the queue manager does not scan the queues looking for expired messages.

To determine the value of this attribute, use the MQIA_EXPIRY_INTERVAL selector with the MQINQ call.

This attribute is supported only on z/OS.

IGQPutAuthority (MQLONG)

Intra-group queuing put authority.

Queue manager – IGQPutAuthority attribute

This attribute is applicable only if the local queue manager is a member of a queue-sharing group. The attribute indicates the type of authority checking that is performed when the local intra-group queuing agent (IGQ agent) removes a message from the shared transmission queue and places the message on a local queue. The value is one of the following:

MQIGQPA_DEFAULT

Default user identifier is used.

The user identifier checked for authorization is the value of the *UserIdentifier* field in the *separate* MQMD that is associated with the message when the message is on the shared transmission queue. This is the user identifier of the program that placed the message on the shared transmission queue, and is usually the same as the user identifier under which the remote queue manager is running.

If the RESLEVEL profile indicates that more than one user identifier is to be checked, the user identifier of the local IGQ agent (*IGQUserId*) is also checked.

MQIGQPA_CONTEXT

Context user identifier is used.

The user identifier checked for authorization is the value of the *UserIdentifier* field in the *separate* MQMD that is associated with the message when the message is on the shared transmission queue. This is the user identifier of the program that placed the message on the shared transmission queue, and is usually the same as the user identifier under which the remote queue manager is running.

If the RESLEVEL profile indicates that more than one user identifier is to be checked, the user identifier of the local IGQ agent (*IGQUserId*) and the value of the *UserIdentifier* field in the *embedded* MQMD are also checked. The latter user identifier is usually the user identifier of the application that originated the message.

MQIGQPA_ONLY_IGQ

Only the IGQ user identifier is used.

The user identifier checked for authorization is the user identifier of the local IGQ agent (*IGQUserId*).

If the RESLEVEL profile indicates that more than one user identifier is to be checked, this user identifier is used for all checks.

MQIGQPA_ALTERNATE_OR_IGQ

Alternate user identifier or IGQ-agent user identifier is used.

The user identifier checked for authorization is the user identifier of the local IGQ agent (*IGQUserId*).

If the RESLEVEL profile indicates that more than one user identifier is to be checked, the value of the *UserIdentifier* field in the *embedded* MQMD is also checked. This user identifier is usually the user identifier of the application that originated the message.

To determine the value of this attribute, use the MQIA_IGQ_PUT_AUTHORITY selector with the MQINQ call.

This attribute is supported only on z/OS.

Queue manager – IGQUserId attribute

IGQUserId (MQLONG)

Intra-group queuing agent user identifier.

This attribute is applicable only if the local queue manager is a member of a queue-sharing group. The attribute specifies the user identifier that is associated with the local intra-group queuing agent (IGQ agent). This identifier is one of the user identifiers that may be checked for authorization when the IGQ agent puts messages on local queues. The actual user identifiers checked depend on the setting of the *IGQPutAuthority* attribute, and on external security options.

If *IGQUserId* is blank, no user identifier is associated with the IGQ agent and the corresponding authorization check is not performed (although other user identifiers may still be checked for authorization).

To determine the value of this attribute, use the MQCA_IGQ_USER_ID selector with the MQINQ call. The length of this attribute is given by MQ_USER_ID_LENGTH.

This attribute is supported only on z/OS.

InhibitEvent (MQLONG)

Controls whether inhibit (Inhibit Get and Inhibit Put) events are generated.

The value is one of the following:

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

For more information about events, see the *WebSphere MQ Event Monitoring* book.

To determine the value of this attribute, use the MQIA_INHIBIT_EVENT selector with the MQINQ call.

- On z/OS, the MQINQ call cannot be used to determine the value of this attribute. You cannot enable event attributes on WebSphere Application Server embedded messaging using reduced function WebSphere MQ.

IntraGroupQueuing (MQLONG)

Intra-group queuing support.

This attribute is applicable only if the local queue manager is a member of a queue-sharing group. The attribute indicates whether intra-group queuing is enabled for the queue-sharing group. The value is one of the following:

MQIGQ_DISABLED

Intra-group queuing disabled.

All messages destined for other queue managers in the queue-sharing group are transmitted using conventional channels.

MQIGQ_ENABLED

Intra-group queuing enabled.

Queue manager – IntraGroupQueuing attribute

Messages destined for other queue managers in the queue-sharing group are transmitted using the shared transmission queue if the following condition is satisfied:

- The length of the message data plus transmission header does not exceed 63 KB (64 512 bytes).

It is recommended that somewhat more space than the size of MQXQH be allocated for the transmission header; the constant MQ_MSG_HEADER_LENGTH is provided for this purpose.

If this condition is not satisfied, the message is transmitted using conventional channels.

Note: When intra-group queuing is enabled, the order of messages transmitted using the shared transmission queue is not preserved relative to those transmitted using conventional channels.

To determine the value of this attribute, use the MQIA_INTRA_GROUP_QUEUING selector with the MQINQ call.

This attribute is supported only on z/OS.

LocalEvent (MQLONG)

Controls whether local error events are generated.

The value is one of the following:

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

For more information about events, see the *WebSphere MQ Event Monitoring* book.

To determine the value of this attribute, use the MQIA_LOCAL_EVENT selector with the MQINQ call.

- On z/OS, the MQINQ call cannot be used to determine the value of this attribute. You cannot enable event attributes on WebSphere Application Server embedded messaging using reduced function WebSphere MQ.

MaxHandles (MQLONG)

Maximum number of handles.

This is the maximum number of open handles that any one task can use concurrently. Each successful MQOPEN call for a single queue (or for an object that is not a queue) uses one handle. That handle becomes available for reuse when the object is closed. However, when a distribution list is opened, each queue in the distribution list is allocated a separate handle, and so that MQOPEN call uses as many handles as there are queues in the distribution list. This must be taken into account when deciding on a suitable value for *MaxHandles*.

The MQPUT1 call performs an MQOPEN call as part of its processing; as a result, MQPUT1 uses as many handles as MQOPEN would, but the handles are used only for the duration of the MQPUT1 call itself.

- On z/OS, *task* means a CICS task, an MVS task, or an IMS dependent region.

Queue manager – MaxHandles attribute

- On Compaq NonStop Kernel, any MaxHandles value specified is ignored.

The value is in the range 1 through 999 999 999. The default value is determined by the environment:

- On z/OS, the default value is 100.
- In all other environments, the default value is 256.

To determine the value of this attribute, use the MQIA_MAX_HANDLES selector with the MQINQ call.

MaxMsgLength (MQLONG)

Maximum message length in bytes.

This is the length of the longest *physical* message that can be handled by the queue manager. However, because the *MaxMsgLength* queue-manager attribute can be set independently of the *MaxMsgLength* queue attribute, the longest physical message that can be placed on a queue is the lesser of those two values.

If the queue manager supports segmentation, it is possible for an application to put a *logical* message that is longer than the lesser of the two *MaxMsgLength* attributes, but only if the application specifies the MQMF_SEGMENTATION_ALLOWED flag in MQMD. If that flag is specified, the upper limit for the length of a logical message is 999 999 999 bytes, but usually resource constraints imposed by the operating system, or by the environment in which the application is running, will result in a lower limit.

The lower limit for the *MaxMsgLength* attribute is 32 KB (32 768 bytes). The upper limit is determined by the environment:

- On AIX, Compaq NonStop Kernel, Compaq OpenVMS Alpha, HP-UX, z/OS, OS/2, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems, the maximum message length is 100 MB (104 857 600 bytes).
- On UNIX systems not listed above, Windows 3.1, Windows 95, Windows 98, plus WebSphere MQ clients connected to these systems, the maximum message length is 4 MB (4 194 304 bytes).

To determine the value of this attribute, use the MQIA_MAX_MSG_LENGTH selector with the MQINQ call.

MaxPriority (MQLONG)

Maximum priority.

This is the maximum message priority supported by the queue manager. Priorities range from zero (lowest) to *MaxPriority* (highest).

To determine the value of this attribute, use the MQIA_MAX_PRIORITY selector with the MQINQ call.

MaxUncommittedMsgs (MQLONG)

Maximum number of uncommitted messages within a unit of work.

On Compaq NonStop Kernel, any MaxUncommittedMsgs value specified is ignored.

Queue manager – MaxUncommittedMsgs attribute

This is the maximum number of uncommitted messages that can exist within a unit of work. The number of uncommitted messages is the sum of the following since the start of the current unit of work:

- Messages put by the application with the MQPMO_SYNCPOINT option
- Messages retrieved by the application with the MQGMO_SYNCPOINT option
- Trigger messages and COA report messages generated by the queue manager for messages put with the MQPMO_SYNCPOINT option
- COD report messages generated by the queue manager for messages retrieved with the MQGMO_SYNCPOINT option

The following are *not* counted as uncommitted messages:

- Messages put or retrieved by the application outside a unit of work
- Trigger messages or COA/COD report messages generated by the queue manager as a result of messages put or retrieved outside a unit of work
- Expiration report messages generated by the queue manager (even if the call causing the expiration report message specified MQGMO_SYNCPOINT)
- Event messages generated by the queue manager (even if the call causing the event message specified MQPMO_SYNCPOINT or MQGMO_SYNCPOINT)

Notes:

1. Exception report messages are generated by the Message Channel Agent (MCA), or by the application, and so are treated in the same way as ordinary messages put or retrieved by the application.
2. When a message or segment is put with the MQPMO_SYNCPOINT option, the number of uncommitted messages is incremented by one regardless of how many physical messages actually result from the put. (More than one physical message may result if the queue manager needs to subdivide the message or segment.)
3. When a distribution list is put with the MQPMO_SYNCPOINT option, the number of uncommitted messages is incremented by one *for each physical message that is generated*. This can be as small as one, or as great as the number of destinations in the distribution list.

The lower limit for this attribute is 1; the upper limit is 999 999 999.

To determine the value of this attribute, use the MQIA_MAX_UNCOMMITTED_MSGS selector with the MQINQ call.

PerformanceEvent (MQLONG)

Controls whether performance-related events are generated.

The value is one of the following:

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

For more information about events, see the *WebSphere MQ Event Monitoring* book.

To determine the value of this attribute, use the MQIA_PERFORMANCE_EVENT selector with the MQINQ call.

Queue manager – PerformanceEvent attribute

- On z/OS, the MQINQ call cannot be used to determine the value of this attribute. You cannot enable event attributes on WebSphere Application Server embedded messaging using reduced function WebSphere MQ.

Platform (MQPLONG)

Platform on which the queue manager is running.

This indicates the operating system on which the queue manager is running:

MQPL_AIX

AIX (same value as MQPL_UNIX).

MQPL_MVS

MVS/ESA (same value as MQPL_ZOS).

MQPL_NSK

Compaq NonStop Kernel.

MQPL_OS2

OS/2.

MQPL_OS390

OS/390 (same value as MQPL_ZOS).

MQPL_OS400

OS/400.

MQPL_UNIX

UNIX systems.

MQPL_VMS

Compaq OpenVMS Alpha.

MQPL_WINDOWS

Windows 3.1.

MQPL_WINDOWS_NT

Windows or Windows 95, Windows 98.

MQPL_ZOS

z/OS.

To determine the value of this attribute, use the MQIA_PLATFORM selector with the MQINQ call.

QMGrDesc (MQCHAR64)

Queue manager description.

This is a field that may be used for descriptive commentary. The content of the field is of no significance to the queue manager, but the queue manager may require that the field contain only characters that can be displayed. It cannot contain any null characters; if necessary, it is padded to the right with blanks. In a DBCS installation, this field can contain DBCS characters (subject to a maximum field length of 64 bytes).

Note: If this field contains characters that are not in the queue manager's character set (as defined by the *CodedCharSetId* queue manager attribute), those characters may be translated incorrectly if this field is sent to another queue manager.

- On z/OS, the default value is the product name and version number.

Queue manager – QMgrDesc attribute

- In all other environments, the default value is blanks.

To determine the value of this attribute, use the MQCA_Q_MGR_DESC selector with the MQINQ call. The length of this attribute is given by MQ_Q_MGR_DESC_LENGTH.

QMgrIdentifier (MQCHAR48)

Unique internally-generated identifier of queue manager.

This is an internally-generated unique name for the queue manager.

To determine the value of this attribute, use the MQCA_Q_MGR_IDENTIFIER selector with the MQINQ call. The length of this attribute is given by MQ_Q_MGR_IDENTIFIER_LENGTH.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

QMgrName (MQCHAR48)

Queue manager name.

This is the name of the local queue manager, that is, the name of the queue manager to which the application is connected.

The first 12 characters of the name are used to construct a unique message identifier (see the *MsgId* field described in Chapter 10, “MQMD – Message descriptor”, on page 141). Queue managers that can intercommunicate must therefore have names that differ in the first 12 characters, in order for message identifiers to be unique in the queue-manager network.

- On z/OS, the name is the same as the subsystem name, which is limited to 4 nonblank characters.

To determine the value of this attribute, use the MQCA_Q_MGR_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_MGR_NAME_LENGTH.

QSGName (MQCHAR4)

Name of queue-sharing group.

This is the name of the queue-sharing group to which the local queue manager belongs. If the local queue manager does not belong to a queue-sharing group, the name is blank.

To determine the value of this attribute, use the MQCA_QSG_NAME selector with the MQINQ call. The length of this attribute is given by MQ_QSG_NAME_LENGTH.

This attribute is supported only on z/OS.

RemoteEvent (MQLONG)

Controls whether remote error events are generated.

The value is one of the following:

Queue manager – RemoteEvent attribute

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

For more information about events, see the *WebSphere MQ Event Monitoring* book.

To determine the value of this attribute, use the MQIA_REMOTE_EVENT selector with the MQINQ call.

- On z/OS, the MQINQ call cannot be used to determine the value of this attribute. You cannot enable event attributes on WebSphere Application Server embedded messaging using reduced function WebSphere MQ.

RepositoryName (MQCHAR48)

Name of cluster for which this queue manager provides repository services.

This is the name of a cluster for which this queue manager provides a repository-manager service. If the queue manager provides this service for more than one cluster, *RepositoryNameList* specifies the name of a namelist object that identifies the clusters, and *RepositoryName* is blank. At least one of *RepositoryName* and *RepositoryNameList* must be blank.

To determine the value of this attribute, use the MQCA_REPOSITORY_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_MGR_NAME_LENGTH.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems. It must be blank on WebSphere Application Server embedded messaging using reduced function WebSphere MQ on z/OS.

RepositoryNameList (MQCHAR48)

Name of namelist object containing names of clusters for which this queue manager provides repository services.

This is the name of a namelist object that contains the names of clusters for which this queue manager provides a repository-manager service. If the queue manager provides this service for only one cluster, the namelist object contains only one name. Alternatively, *RepositoryName* can be used to specify the name of the cluster, in which case *RepositoryNameList* is blank. At least one of *RepositoryName* and *RepositoryNameList* must be blank.

To determine the value of this attribute, use the MQCA_REPOSITORY_NAMELIST selector with the MQINQ call. The length of this attribute is given by MQ_NAMELIST_NAME_LENGTH.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, OS/2, Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/400, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems. It must be blank on WebSphere Application Server embedded messaging using reduced function WebSphere MQ on z/OS.

StartStopEvent (MQLONG)

Controls whether start and stop events are generated.

The value is one of the following:

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

For more information about events, see the *WebSphere MQ Event Monitoring* book.

To determine the value of this attribute, use the MQIA_START_STOP_EVENT selector with the MQINQ call.

- On z/OS, the MQINQ call cannot be used to determine the value of this attribute. You cannot enable event attributes on WebSphere Application Server embedded messaging using reduced function WebSphere MQ.

SyncPoint (MQLONG)

Syncpoint availability.

This indicates whether the local queue manager supports units of work and syncpointing with the MQGET, MQPUT, and MQPUT1 calls.

MQSP_AVAILABLE

Units of work and syncpointing available.

MQSP_NOT_AVAILABLE

Units of work and syncpointing not available.

- On z/OS and Compaq NonStop Kernel, this value is never returned.

To determine the value of this attribute, use the MQIA_SYNCPOINT selector with the MQINQ call.

TriggerInterval (MQLONG)

Trigger-message interval.

This is a time interval (in milliseconds) used to restrict the number of trigger messages. This is relevant only when the *TriggerType* is MQTT_FIRST. In this case trigger messages are normally generated only when a suitable message arrives on the queue, and the queue was previously empty. Under certain circumstances, however, an additional trigger message can be generated with MQTT_FIRST triggering even if the queue was not empty. These additional trigger messages are not generated more often than every *TriggerInterval* milliseconds.

For more information on triggering, see the *WebSphere MQ Application Programming Guide*.

The value is not less than 0 and not greater than 999 999 999. The default value is 999 999 999.

To determine the value of this attribute, use the MQIA_TRIGGER_INTERVAL selector with the MQINQ call.

Object attributes

Chapter 44. Attributes for authentication information objects

Authentication information objects are supported in the following environments: AIX, HP-UX, z/OS, OS/400, Solaris, Linux, and Windows.

The following table summarizes the attributes that are specific to authentication information objects. The attributes are described in alphabetic order.

Note: The names of the attributes shown in this book are the names used with the MQINQ and MQSET calls. When MQSC commands are used to define, alter, or display attributes, alternative short names are used; see the *WebSphere MQ Script (MQSC) Command Reference* for details.

Table 98. Attributes for process definitions

Attribute	Description	Page
<i>AlterationDate</i>	Date when definition was last changed	521
<i>AlterationTime</i>	Time when definition was last changed	521
<i>AuthInfoConnName</i>	The DNS name or IP address of the host on which the LDAP server is running, with an optional port number. This keyword is required.	522
<i>AuthInfoDesc</i>	Plain-text comment. It provides descriptive information about the authentication information object when an operator issues the DISPLAY AUTHINFO command.	522
<i>AuthInfoName</i>	Name of the authentication information object.	522
<i>AuthInfoType</i>	The type of authentication information.	522
<i>LDAPPassword</i>	The password associated with the Distinguished Name of the user who is accessing the LDAP server.	522
<i>LDAPUserName</i>	The Distinguished Name of the user who is accessing the LDAP server.	522

Attribute descriptions

An authentication information object has the attributes described below.

AlterationDate (MQCHAR12)

Date when definition was last changed.

This is the date when the definition was last changed. The format of the date is YYYY-MM-DD, padded with two trailing blanks to make the length 12 bytes.

AlterationTime (MQCHAR8)

Time when definition was last changed.

This is the time when the definition was last changed. The format of the time is HH.MM.SS using the 24-hour clock, with a leading zero if the hour is less than 10 (for example 09.10.20).

Authentication information – AlterationTime attribute

- On z/OS, the time is Greenwich Mean Time (GMT), subject to the system clock being set accurately to GMT.
- In other environments, the time is local time.

AuthInfoConnName (MQCHAR264)

The DNS name or IP address of the host on which the LDAP CRL server is running, with an optional port number. This keyword is required.

The syntax for CONNAME is the same as for channels. For example,
`conname('hostname(nnn)')`

where *hostname* is the DNS name or IP address and where *nnn* is the port number. If *nnn* is not provided, the default port number 389 is used.

The maximum length for the field is 264 characters on OS/400, UNIX systems, and Windows and 48 characters on z/OS.

AuthInfoDesc (MQCHAR64)

Plain-text comment. It provides descriptive information about the authentication information object when an operator issues the DISPLAY AUTHINFO command.

It should contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

AuthInfoName (MQCHAR48)

Name of the authentication information object.

The name must not be the same as any other authentication information object name currently defined on this queue manager (unless REPLACE or ALTER is specified).

AuthInfoType (MQLONG)

The type of authentication information. The value must be CRLLDAP, meaning that Certificate Revocation List checking is done using LDAP servers.

LDAPPassword (MQCHAR32)

The password associated with the Distinguished Name of the user who is accessing the LDAP CRL server.

Its maximum size is 32 characters. The default value is blank.

LDAPUserName (MQ_DISTINGUISHED_NAME_LENGTH)

The Distinguished Name of the user who is accessing the LDAP CRL server.

The maximum size for the user name is 1024 characters on OS/400, UNIX systems, and Windows, and 256 characters on z/OS.

Authentication information – LDAPUserName attribute

| The maximum accepted line length is defined to be BUFSIZ, which can be found in
| stdio.h.

| If you use asterisks (*) in the user name they are treated as literal characters, and
| not as wild cards, because LDAPUserName is a specific name and not a string
| used for matching.

Object attributes

Part 4. Appendixes

Appendix A. Return codes

This book introduces the return codes associated with the MQI and MQAI. For each call, a **completion** code and a **reason** code are returned by the queue manager or by an exit routine, to indicate the success or failure of the call.

Applications must not depend upon errors being checked for in a specific order, except where specifically noted. If more than one completion code or reason code could arise from a call, the particular error reported depends on the implementation.

Completion codes

The completion code parameter (*CompCode*) allows the caller to see quickly whether the call completed successfully, completed partially, or failed.

The following is a list of completion codes, with more detail than is given in the call descriptions:

MQCC_OK

Successful completion.

The call completed fully; all output parameters have been set. The *Reason* parameter always has the value MQRC_NONE in this case.

MQCC_WARNING

Warning (partial completion).

The call completed partially. Some output parameters may have been set in addition to the *CompCode* and *Reason* output parameters. The *Reason* parameter gives additional information about the partial completion.

MQCC_FAILED

Call failed.

The processing of the call did not complete, and the state of the queue manager is normally unchanged; exceptions are specifically noted. The *CompCode* and *Reason* output parameters have been set; other parameters are unchanged, except where noted.

The reason may be a fault in the application program, or it may be a result of some situation external to the program, for example the user's authority may have been revoked. The *Reason* parameter gives additional information about the error.

Return codes

Reason codes

The reason code parameter (*Reason*) is a qualification to the completion code parameter (*CompCode*).

If there is no special reason to report, MQRC_NONE is returned. A successful call returns MQCC_OK and MQRC_NONE.

If the completion code is either MQCC_WARNING or MQCC_FAILED, the queue manager always reports a qualifying reason; details are given under each call description.

Where user exit routines set completion codes and reasons, they should adhere to these rules. In addition, any special reason values defined by user exits should be less than zero, to ensure that they do not conflict with values defined by the queue manager. Exits can set reasons already defined by the queue manager, where these are appropriate.

Reason codes also occur in:

- The *Reason* field of the MQDLH structure
- The *Feedback* field of the MQMD structure

For complete descriptions of reason codes see:

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

Appendix B. MQ constants

This appendix specifies the values of the named constants that apply to the main Message Queue Interface (MQI). This information is general-use programming interface information.

The constants are grouped according to the parameter or field to which they relate. All of the names of the constants in a group begin with a common prefix of the form "MQxxxxx_", where xxxxx represents a string of 0 through 5 characters that indicates the parameter or field to which the values relate. The constants are ordered alphabetically by this prefix.

Notes:

1. For constants with numeric values, the values are shown in both decimal and hexadecimal forms.
2. Hexadecimal values are represented using the notation X'hhhh', where each "h" denotes a single hexadecimal digit.
3. Character values are shown delimited by single quotation marks; the quotation marks are not part of the value.
4. Blanks in character values are represented by one or more occurrences of the symbol "b".
5. If the value is shown as "(variable)", it indicates that the value of the constant depends on the environment in which the application is running.

List of constants

The following sections list all of the named constants that are mentioned in this book, and show their values.

MQ_* (Lengths of character string and byte fields)

See the *CharAttrs* parameter described in Chapter 35, "MQINQ – Inquire object attributes", on page 393 and Chapter 35, "MQINQ – Inquire object attributes", on page 393.

MQ_ABEND_CODE_LENGTH	4	X'00000004'
MQ_ACCOUNTING_TOKEN_LENGTH	32	X'00000020'
MQ_APPL_IDENTITY_DATA_LENGTH	32	X'00000020'
MQ_APPL_ORIGIN_DATA_LENGTH	4	X'00000004'
MQ_ATTENTION_ID_LENGTH	4	X'00000004'
MQ_AUTHENTICATOR_LENGTH	8	X'00000008'
MQ_CANCEL_CODE_LENGTH	4	X'00000004'
MQ_CF_STRUC_NAME_LENGTH	12	X'0000000C'
MQ_CLUSTER_NAME_LENGTH	48	X'00000030'
MQ_CONN_TAG_LENGTH	128	X'00000080'
MQ_CORREL_ID_LENGTH	24	X'00000018'
MQ_CREATION_DATE_LENGTH	12	X'0000000C'
MQ_CREATION_TIME_LENGTH	8	X'00000008'
MQ_DATE_LENGTH	12	X'0000000C'
MQ_EXIT_NAME_LENGTH	(variable)	
MQ_FACILITY_LENGTH	8	X'00000008'
MQ_FACILITY_LIKE_LENGTH	4	X'00000004'

MQACTT_* (Accounting token type)

See the *AccountingToken* field described in Chapter 10, “MQMD – Message descriptor”, on page 141.

MQACTT_UNKNOWN	X'00'
MQACTT_CICS_LUOW_ID	X'01'
MQACTT_OS2_DEFAULT	X'04'
MQACTT_DOS_DEFAULT	X'05'
MQACTT_UNIX_NUMERIC_ID	X'06'
MQACTT_OS400_ACCOUNT_TOKEN	X'08'
MQACTT_WINDOWS_DEFAULT	X'09'
MQACTT_NT_SECURITY_ID	X'0B'
MQACTT_USER	X'19'

MQAIR_* (Authentication information record structure identifier)

See the *StrucId* field described in Chapter 2, “MQAIR – Authentication information record”, on page 31.

MQAIR_STRUC_ID	'AIRb'
----------------	--------

For the C programming language, the following array version is also defined:

MQAIR_STRUC_ID_ARRAY	'A','I','R','b'
----------------------	-----------------

MQAIR_* (Authentication information record version)

See the *Version* field described in Chapter 2, “MQAIR – Authentication information record”, on page 31.

MQAIR_VERSION_1	1	X'00000001'
MQAIR_CURRENT_VERSION	(variable)	

MQAIT_* (Authentication information type)

See the *AuthInfoType* field described in Chapter 2, “MQAIR – Authentication information record”, on page 31.

MQAIT_CRL_LDAP	1	X'00000001'
----------------	---	-------------

MQAT_* (Application type)

See the *PutApplType* field described in Chapter 10, “MQMD – Message descriptor”, on page 141, and the *ApplType* attribute described in Chapter 42, “Attributes for process definitions”, on page 495.

MQ constants

MQAT_UNKNOWN	-1	X'FFFFFFFF'
MQAT_NO_CONTEXT	0	X'00000000'
MQAT_CICS	1	X'00000001'
MQAT_MVS	2	X'00000002'
MQAT_OS390	2	X'00000002'
MQAT_ZOS	2	X'00000002'
MQAT_IMS	3	X'00000003'
MQAT_OS2	4	X'00000004'
MQAT_DOS	5	X'00000005'
MQAT_AIX	6	X'00000006'
MQAT_UNIX	6	X'00000006'
MQAT_QMGR	7	X'00000007'
MQAT_OS400	8	X'00000008'
MQAT_WINDOWS	9	X'00000009'
MQAT_CICS_VSE	10	X'0000000A'
MQAT_WINDOWS_NT	11	X'0000000B'
MQAT_VMS	12	X'0000000C'
MQAT_GUARDIAN	13	X'0000000D'
MQAT_NSK	13	X'0000000D'
MQAT_VOS	14	X'0000000E'
MQAT_IMS_BRIDGE	19	X'00000013'
MQAT_XCF	20	X'00000014'
MQAT_CICS_BRIDGE	21	X'00000015'
MQAT_NOTES_AGENT	22	X'00000016'
MQAT_BROKER	26	X'0000001A'
MQAT_JAVA	28	X'0000001C'
MQAT_DQM	29	X'0000001D'
MQAT_WLM	31	X'0000001F'
MQAT_USER_FIRST	65536	X'00010000'
MQAT_USER_LAST	99999999	X'3B9AC9FF'
MQAT_DEFAULT	(variable)	

MQBND_* (Binding)

See the *DefBind* attribute described in Chapter 40, “Attributes for queues”, on page 457.

MQBND_BIND_ON_OPEN	0	X'00000000'
MQBND_BIND_NOT_FIXED	1	X'00000001'

MQBO_* (Begin options)

See the *Options* field described in Chapter 3, “MQBO – Begin options”, on page 37.

MQBO_NONE	0	X'00000000'
-----------	---	-------------

MQBO_* (Begin options structure identifier)

See the *StrucId* field described in Chapter 3, “MQBO – Begin options”, on page 37.

MQBO_STRUC_ID	'B0bb'
---------------	--------

For the C programming language, the following array version is also defined:

```
MQBO_STRUC_ID_ARRAY          'B','0','b','b'
```

MQBO_* (Begin options version)

See the *Version* field described in Chapter 3, “MQBO – Begin options”, on page 37.

```
MQBO_VERSION_1              1          X'00000001'
MQBO_CURRENT_VERSION        1          X'00000001'
```

MQCA_* (Character attribute selector)

See the *Selectors* parameter described in Chapter 35, “MQINQ – Inquire object attributes”, on page 393 and Chapter 35, “MQINQ – Inquire object attributes”, on page 393.

```
MQCA_FIRST                  2001      X'000007D1'
MQCA_APPL_ID                2001      X'000007D1'
MQCA_BASE_Q_NAME           2002      X'000007D2'
MQCA_COMMAND_INPUT_Q_NAME 2003      X'000007D3'
MQCA_CREATION_DATE         2004      X'000007D4'
MQCA_CREATION_TIME         2005      X'000007D5'
MQCA_DEAD_LETTER_Q_NAME    2006      X'000007D6'
MQCA_ENV_DATA              2007      X'000007D7'
MQCA_INITIATION_Q_NAME     2008      X'000007D8'
MQCA_NAMELIST_DESC         2009      X'000007D9'
MQCA_NAMELIST_NAME         2010      X'000007DA'
MQCA_PROCESS_DESC          2011      X'000007DB'
MQCA_PROCESS_NAME          2012      X'000007DC'
MQCA_Q_DESC                2013      X'000007DD'
MQCA_Q_MGR_DESC            2014      X'000007DE'
MQCA_Q_MGR_NAME            2015      X'000007DF'
MQCA_Q_NAME                2016      X'000007E0'
MQCA_REMOTE_Q_MGR_NAME     2017      X'000007E1'
MQCA_REMOTE_Q_NAME         2018      X'000007E2'
MQCA_BACKOUT_REQ_Q_NAME    2019      X'000007E3'
MQCA_NAMES                 2020      X'000007E4'
MQCA_USER_DATA             2021      X'000007E5'
MQCA_STORAGE_CLASS         2022      X'000007E6'
MQCA_TRIGGER_DATA          2023      X'000007E7'
MQCA_XMIT_Q_NAME           2024      X'000007E8'
MQCA_DEF_XMIT_Q_NAME       2025      X'000007E9'
MQCA_CHANNEL_AUTO_DEF_EXIT 2026      X'000007EA'
MQCA_ALTERATION_DATE       2027      X'000007EB'
MQCA_ALTERATION_TIME       2028      X'000007EC'
MQCA_CLUSTER_NAME          2029      X'000007ED'
MQCA_CLUSTER_NAMELIST      2030      X'000007EE'
MQCA_CLUSTER_Q_MGR_NAME    2031      X'000007EF'
MQCA_Q_MGR_IDENTIFIER       2032      X'000007F0'
MQCA_CLUSTER_WORKLOAD_EXIT 2033      X'000007F1'
MQCA_CLUSTER_WORKLOAD_DATA 2034      X'000007F2'
MQCA_REPOSITORY_NAME       2035      X'000007F3'
```

MQ constants

MQCA_REPOSITORY_NAMELIST	2036	X'000007F4'
MQCA_CLUSTER_DATE	2037	X'000007F5'
MQCA_CLUSTER_TIME	2038	X'000007F6'
MQCA_CF_STRUC_NAME	2039	X'000007F7'
MQCA_QSG_NAME	2040	X'000007F8'
MQCA_IGQ_USER_ID	2041	X'000007F9'
MQCA_USER_LIST	4000	X'00000FA0'
MQCA_LAST	4000	X'00000FA0'
MQCA_LAST_USED	(variable)	

MQCADSD_* (CICS header ADS descriptor)

See the *ADSDescriptor* field described in Chapter 4, "MQCIH – CICS bridge header", on page 41.

MQCADSD_NONE	0	X'00000000'
MQCADSD_SEND	1	X'00000001'
MQCADSD_RECV	16	X'00000010'
MQCADSD_MSGFORMAT	256	X'00000100'

MQCC_* (Completion code)

See the *CompCode* parameter described in each MQI call.

MQCC_OK	0	X'00000000'
MQCC_WARNING	1	X'00000001'
MQCC_FAILED	2	X'00000002'

MQCCSI_* (Coded character set identifier)

See the *CodedCharSetId* field described in Chapter 10, "MQMD – Message descriptor", on page 141 and in other structures.

MQCCSI_INHERIT	-2	X'FFFFFFFFE'
MQCCSI_EMBEDDED	-1	X'FFFFFFFF'
MQCCSI_UNDEFINED	0	X'00000000'
MQCCSI_DEFAULT	0	X'00000000'
MQCCSI_Q_MGR	0	X'00000000'

MQCCT_* (CICS header conversational task)

See the *ConversationalTask* field described in Chapter 4, "MQCIH – CICS bridge header", on page 41.

MQCCT_NO	0	X'00000000'
MQCCT_YES	1	X'00000001'

MQ constants

MQCI_* (Correlation identifier)

See the *CorrelId* field described in Chapter 10, “MQMD – Message descriptor”, on page 141.

MQCI_NONE	X'00...00' (24 nulls)
MQCI_NEW_SESSION	X'414D51214E45575F53455353'
	X'494F4E5F434F5252454C4944'

For the C programming language, the following array versions are also defined:

MQCI_NONE_ARRAY	'\0', '\0', ... '\0', '\0'
MQCI_NEW_SESSION_ARRAY	'\x41', '\x4d', '\x51', '\x21', '\x4e', '\x45', '\x57', '\x5f', '\x53', '\x45', '\x53', '\x53', '\x49', '\x4f', '\x4e', '\x5f', '\x43', '\x4f', '\x52', '\x52', '\x45', '\x4c', '\x49', '\x44'

MQCIH_* (CICS header flags)

See the *Flags* field described in Chapter 4, “MQCIH – CICS bridge header”, on page 41.

	MQCIH_NONE	0	X'00000000'
	MQCIH_NO_SYNC_ON_RETURN	0	X'00000000'
	MQCIH_REPLY_WITH_NULLS	0	X'00000000'
	MQCIH_UNLIMITED_EXPIRATION	0	X'00000000'
	MQCIH_PASS_EXPIRATION	1	X'00000001'
	MQCIH_REPLY_WITHOUT_NULLS	2	X'00000002'
	MQCIH_SYNC_ON_RETURN	4	X'00000004'

MQCIH_* (CICS header length)

See the *StrucLength* field described in Chapter 4, “MQCIH – CICS bridge header”, on page 41.

	MQCIH_LENGTH_1	164	X'000000A4'
	MQCIH_LENGTH_2	180	X'000000B4'
	MQCIH_CURRENT_LENGTH	180	X'000000B4'

MQCIH_* (CICS header structure identifier)

See the *StrucId* field described in Chapter 4, “MQCIH – CICS bridge header”, on page 41.

MQCIH_STRUC_ID	'CIHb'
----------------	--------

For the C programming language, the following array version is also defined:

MQCIH_STRUC_ID_ARRAY	'C', 'I', 'H', 'b'
----------------------	--------------------

MQCIH_* (CICS header version)

See the *Version* field described in Chapter 4, “MQCIH – CICS bridge header”, on page 41.

MQCIH_VERSION_1	1	X'00000001'
MQCIH_VERSION_2	2	X'00000002'
MQCIH_CURRENT_VERSION	2	X'00000002'

MQCLT_* (CICS header link type)

See the *LinkType* field described in Chapter 4, “MQCIH – CICS bridge header”, on page 41.

MQCLT_PROGRAM	1	X'00000001'
MQCLT_TRANSACTION	2	X'00000002'

MQCMDL_* (Command level)

See the *CommandLevel* attribute described in Chapter 43, “Attributes for the queue manager”, on page 501.

MQCMDL_LEVEL_1	100	X'00000064'
MQCMDL_LEVEL_101	101	X'00000065'
MQCMDL_LEVEL_110	110	X'0000006E'
MQCMDL_LEVEL_114	114	X'00000072'
MQCMDL_LEVEL_120	120	X'00000078'
MQCMDL_LEVEL_200	200	X'000000C8'
MQCMDL_LEVEL_201	201	X'000000C9'
MQCMDL_LEVEL_210	210	X'000000D2'
MQCMDL_LEVEL_220	220	X'000000DC'
MQCMDL_LEVEL_221	221	X'000000DD'
MQCMDL_LEVEL_320	320	X'00000140'
MQCMDL_LEVEL_420	420	X'000001A4'
MQCMDL_LEVEL_500	500	X'000001F4'
MQCMDL_LEVEL_510	510	X'000001FE'
MQCMDL_LEVEL_520	520	X'00000208'
MQCMDL_LEVEL_530	530	X'00000212'

MQCNO_* (Connect options)

See the *Options* field described in Chapter 5, “MQCNO – Connect options”, on page 61.

MQCNO_STANDARD_BINDING	0	X'00000000'
MQCNO_FASTPATH_BINDING	1	X'00000001'
MQCNO_SERIALIZE_CONN_TAG_Q_MGR	2	X'00000002'
MQCNO_SERIALIZE_CONN_TAG_QSG	4	X'00000004'
MQCNO_RESTRICT_CONN_TAG_Q_MGR	8	X'00000008'
MQCNO_RESTRICT_CONN_TAG_QSG	16	X'00000010'
MQCNO_HANDLE_SHARE_NONE	32	X'00000020'
MQCNO_HANDLE_SHARE_BLOCK	64	X'00000040'
MQCNO_HANDLE_SHARE_NO_BLOCK	128	X'00000080'
MQCNO_NONE	0	X'00000000'

MQ constants

MQCRC_APPLICATION_ABEND	5	X'00000005'
MQCRC_SECURITY_ERROR	6	X'00000006'
MQCRC_PROGRAM_NOT_AVAILABLE	7	X'00000007'
MQCRC_BRIDGE_TIMEOUT	8	X'00000008'
MQCRC_TRANSID_NOT_AVAILABLE	9	X'00000009'

MQCSC_* (CICS header transaction start code)

See the *StartCode* field described in Chapter 4, “MQCIH – CICS bridge header”, on page 41.

MQCSC_START	'Sbbb'
MQCSC_STARTDATA	'SDbbb'
MQCSC_TERMINPUT	'Tbbb'
MQCSC_NONE	'bbbb'

For the C programming language, the following array versions are also defined:

MQCSC_START_ARRAY	'S','b','b','b'
MQCSC_STARTDATA_ARRAY	'S','D','b','b'
MQCSC_TERMINPUT_ARRAY	'T','D','b','b'
MQCSC_NONE_ARRAY	'b','b','b','b'

MQCT_* (Connection tag)

See the *ConnTag* field described in Chapter 5, “MQCNO – Connect options”, on page 61.

MQCT_NONE	X'00...00' (128 nulls)
-----------	------------------------

For the C programming language, the following array version is also defined:

MQCT_NONE_ARRAY	'\0','\0',...'\0','\0'
-----------------	------------------------

MQCTES_* (CICS header task end status)

See the *TaskEndStatus* field described in Chapter 4, “MQCIH – CICS bridge header”, on page 41.

MQCTES_NOSYNC	0	X'00000000'
MQCTES_COMMIT	256	X'00000100'
MQCTES_BACKOUT	4352	X'00001100'
MQCTES_ENDTASK	65536	X'00010000'

MQCUOWC_* (CICS header unit-of-work control)

See the *UOWControl* field described in Chapter 4, “MQCIH – CICS bridge header”, on page 41.

MQ constants

MQCUOWC_MIDDLE	16	X'00000010'
MQCUOWC_FIRST	17	X'00000011'
MQCUOWC_COMMIT	256	X'00000100'
MQCUOWC_LAST	272	X'00000110'
MQCUOWC_ONLY	273	X'00000111'
MQCUOWC_BACKOUT	4352	X'00001100'
MQCUOWC_CONTINUE	65536	X'00010000'

MQDCC_* (Convert-characters masks and factors)

See the *Options* parameter described in “MQXCNV – Convert characters” on page 595.

MQDCC_SOURCE_ENC_MASK	240	X'000000F0'
MQDCC_TARGET_ENC_MASK	3840	X'000000F00'
MQDCC_SOURCE_ENC_FACTOR	16	X'00000010'
MQDCC_TARGET_ENC_FACTOR	256	X'00000100'

MQDCC_* (Convert-characters options)

See the *Options* parameter described in “MQXCNV – Convert characters” on page 595.

MQDCC_SOURCE_ENC_UNDEFINED	0	X'00000000'
MQDCC_TARGET_ENC_UNDEFINED	0	X'00000000'
MQDCC_NONE	0	X'00000000'
MQDCC_DEFAULT_CONVERSION	1	X'00000001'
MQDCC_FILL_TARGET_BUFFER	2	X'00000002'
MQDCC_SOURCE_ENC_NORMAL	16	X'00000010'
MQDCC_SOURCE_ENC_REVERSED	32	X'00000020'
MQDCC_TARGET_ENC_NORMAL	256	X'00000100'
MQDCC_TARGET_ENC_REVERSED	512	X'00000200'
MQDCC_SOURCE_ENC_NATIVE	(variable)	
MQDCC_TARGET_ENC_NATIVE	(variable)	

MQDH_* (Distribution header structure identifier)

See the *StrucId* field described in Chapter 6, “MQDH – Distribution header”, on page 75.

```
MQDH_STRUC_ID          'DHbb'
```

For the C programming language, the following array version is also defined:

```
MQDH_STRUC_ID_ARRAY    'D','H','b','b'
```

MQDH_* (Distribution header version)

See the *Version* field described in Chapter 6, “MQDH – Distribution header”, on page 75.

```
MQDH_VERSION_1        1          X'00000001'
```

MQDH_CURRENT_VERSION	1	X'00000001'
----------------------	---	-------------

MQDHF_* (Distribution header flags)

See the *Flags* field described in Chapter 6, “MQDH – Distribution header”, on page 75.

MQDHF_NONE	0	X'00000000'
MQDHF_NEW_MSG_IDS	1	X'00000001'

MQDL_* (Distribution list support)

See the *DistLists* attributes described in Chapter 43, “Attributes for the queue manager”, on page 501 and Chapter 40, “Attributes for queues”, on page 457.

MQDL_NOT_SUPPORTED	0	X'00000000'
MQDL_SUPPORTED	1	X'00000001'

MQDLH_* (Dead-letter header structure identifier)

See the *StrucId* field described in Chapter 7, “MQDLH – Dead-letter header”, on page 83.

MQDLH_STRUC_ID	'DLHb'
----------------	--------

For the C programming language, the following array version is also defined:

MQDLH_STRUC_ID_ARRAY	'D','L','H','b'
----------------------	-----------------

MQDLH_* (Dead-letter header version)

See the *Version* field described in Chapter 7, “MQDLH – Dead-letter header”, on page 83.

MQDLH_VERSION_1	1	X'00000001'
MQDLH_CURRENT_VERSION	1	X'00000001'

MQDXP_* (Data-conversion-exit parameter structure identifier)

See the *StrucId* field described in “MQDXP – Data-conversion exit parameter” on page 589.

MQDXP_STRUC_ID	'DXPb'
----------------	--------

For the C programming language, the following array version is also defined:

MQDXP_STRUC_ID_ARRAY	'D','X','P','b'
----------------------	-----------------

MQ constants

MQDXP_* (Data-conversion-exit parameter structure version)

See the *Version* field described in “MQDXP – Data-conversion exit parameter” on page 589.

MQDXP_VERSION_1	1	X'00000001'
MQDXP_CURRENT_VERSION	1	X'00000001'

MQEC_* (Signal event-control-block completion code)

See the *Signal1* field described in Chapter 8, “MQGMO – Get-message options”, on page 95.

MQEC_MSG_ARRIVED	2	X'00000002'
MQEC_WAIT_INTERVAL_EXPIRED	3	X'00000003'
MQEC_WAIT_CANCELED	4	X'00000004'
MQEC_Q_MGR QUIESCING	5	X'00000005'
MQEC_CONNECTION QUIESCING	6	X'00000006'

MQEI_* (Expiry interval)

See the *Expiry* field described in Chapter 10, “MQMD – Message descriptor”, on page 141.

MQEI_UNLIMITED	-1	X'FFFFFFFF'
----------------	----	-------------

MQENC_* (Encoding)

See the *Encoding* field described in Chapter 10, “MQMD – Message descriptor”, on page 141.

MQENC_NATIVE	(variable)	
--------------	------------	--

This constant has the following values in the environments indicated:

	Compaq NonStop Kernel	273	X'00000111'
	Compaq OpenVMS Alpha	273	X'00000111'
	Linux for Intel	546	X'00000222'
	OS/400	273	X'00000111'
	Solaris	273	X'00000111'
	UNIX systems (AIX, AT&T, HP-UX)	273	X'00000111'
	Windows, OS/2	546	X'00000222'
	Micro Focus COBOL on Windows, OS/2	17	X'00000011'
	z/OS	785	X'00000311'

MQENC_* (Encoding masks)

See Appendix D, “Machine encodings”, on page 571.

MQENC_INTEGER_MASK	15	X'0000000F'
MQENC_DECIMAL_MASK	240	X'000000F0'
MQENC_FLOAT_MASK	3840	X'00000F00'

MQENC_RESERVED_MASK	-4096	X'FFFFFF00'
---------------------	-------	-------------

MQENC_* (Encoding for packed-decimal integers)

See Appendix D, "Machine encodings", on page 571.

MQENC_DECIMAL_UNDEFINED	0	X'00000000'
MQENC_DECIMAL_NORMAL	16	X'00000010'
MQENC_DECIMAL_REVERSED	32	X'00000020'

MQENC_* (Encoding for floating-point numbers)

See Appendix D, "Machine encodings", on page 571.

MQENC_FLOAT_UNDEFINED	0	X'00000000'
MQENC_FLOAT_IEEE_NORMAL	256	X'00000100'
MQENC_FLOAT_IEEE_REVERSED	512	X'00000200'
MQENC_FLOAT_S390	768	X'00000300'

MQENC_* (Encoding for binary integers)

See Appendix D, "Machine encodings", on page 571.

MQENC_INTEGER_UNDEFINED	0	X'00000000'
MQENC_INTEGER_NORMAL	1	X'00000001'
MQENC_INTEGER_REVERSED	2	X'00000002'

MQEVR_* (Event reporting)

See the *QDepthHighEvent*, *QDepthLowEvent*, and *QDepthMaxEvent* attributes described in Chapter 40, "Attributes for queues", on page 457, and the *AuthorityEvent*, *ChannelAutoDefEvent*, *InhibitEvent*, *LocalEvent*, *PerformanceEvent*, *RemoteEvent*, and *StartStopEvent* attributes described in Chapter 43, "Attributes for the queue manager", on page 501.

MQEVR_DISABLED	0	X'00000000'
MQEVR_ENABLED	1	X'00000001'

MQEXPI_* (Expiry scan interval)

See the *ExpiryInterval* attribute described in Chapter 43, "Attributes for the queue manager", on page 501.

MQEXPI_OFF	0	X'00000000'
------------	---	-------------

MQ constants

MQFB_* (Feedback)

See the *Feedback* field described in Chapter 10, “MQMD – Message descriptor”, on page 141, and the *Reason* field described in Chapter 7, “MQDLH – Dead-letter header”, on page 83; see also the MQRC_* values.

MQFB_NONE	0	X'00000000'
MQFB_SYSTEM_FIRST	1	X'00000001'
MQFB_QUIT	256	X'00000100'
MQFB_EXPIRATION	258	X'00000102'
MQFB_COA	259	X'00000103'
MQFB_COD	260	X'00000104'
MQFB_CHANNEL_COMPLETED	262	X'00000106'
MQFB_CHANNEL_FAIL_RETRY	263	X'00000107'
MQFB_CHANNEL_FAIL	264	X'00000108'
MQFB_APPL_CANNOT_BE_STARTED	265	X'00000109'
MQFB_TM_ERROR	266	X'0000010A'
MQFB_APPL_TYPE_ERROR	267	X'0000010B'
MQFB_STOPPED_BY_MSG_EXIT	268	X'0000010C'
MQFB_XMIT_Q_MSG_ERROR	271	X'0000010F'
MQFB_PAN	275	X'00000113'
MQFB_NAN	276	X'00000114'
MQFB_STOPPED_BY_CHAD_EXIT	277	X'00000115'
MQFB_STOPPED_BY_PUBSUB_EXIT	279	X'00000117'
MQFB_NOT_A_REPOSITORY_MSG	280	X'00000118'
MQFB_BIND_OPEN_CLUSRCVR_DEL	281	X'00000119'
MQFB_DATA_LENGTH_ZERO	291	X'00000123'
MQFB_DATA_LENGTH_NEGATIVE	292	X'00000124'
MQFB_DATA_LENGTH_TOO_BIG	293	X'00000125'
MQFB_BUFFER_OVERFLOW	294	X'00000126'
MQFB_LENGTH_OFF_BY_ONE	295	X'00000127'
MQFB_IIH_ERROR	296	X'00000128'
MQFB_NOT_AUTHORIZED_FOR_IMS	298	X'0000012A'
MQFB_IMS_ERROR	300	X'0000012C'
MQFB_IMS_FIRST	301	X'0000012D'
MQFB_IMS_LAST	399	X'0000018F'
MQFB_CICS_INTERNAL_ERROR	401	X'00000191'
MQFB_CICS_NOT_AUTHORIZED	402	X'00000192'
MQFB_CICS_BRIDGE_FAILURE	403	X'00000193'
MQFB_CICS_CORREL_ID_ERROR	404	X'00000194'
MQFB_CICS_CCSID_ERROR	405	X'00000195'
MQFB_CICS_ENCODING_ERROR	406	X'00000196'
MQFB_CICS_CIH_ERROR	407	X'00000197'
MQFB_CICS_UOW_ERROR	408	X'00000198'
MQFB_CICS_COMMAREA_ERROR	409	X'00000199'
MQFB_CICS_APPL_NOT_STARTED	410	X'0000019A'
MQFB_CICS_APPL_ABENDED	411	X'0000019B'
MQFB_CICS_DLQ_ERROR	412	X'0000019C'
MQFB_CICS_UOW_BACKED_OUT	413	X'0000019D'
MQFB_SYSTEM_LAST	65535	X'0000FFFF'
MQFB_APPL_FIRST	65536	X'00010000'
MQFB_APPL_LAST	99999999	X'3B9AC9FF'

MQFMT_* (Format)

See the *Format* field described in Chapter 10, “MQMD – Message descriptor”, on page 141 and in other structures.

MQFMT_NONE	'bbbbbbbbb'
MQFMT_ADMIN	'MQADMINb'
MQFMT_CHANNEL_COMPLETED	'MQCHCOMb'
MQFMT_CICS	'MQCICSbb'
MQFMT_COMMAND_1	'MQCMD1bb'
MQFMT_COMMAND_2	'MQCMD2bb'
MQFMT_DEAD_LETTER_HEADER	'MQDEADbb'
MQFMT_DIST_HEADER	'MQHDISTb'
MQFMT_EVENT	'MQEVENTb'
MQFMT_IMS	'MQIMSbbb'
MQFMT_IMS_VAR_STRING	'MQIMSVSb'
MQFMT_MD_EXTENSION	'MQHMDEbb'
MQFMT_PCF	'MQPCFbbb'
MQFMT_REF_MSG_HEADER	'MQHREFbb'
MQFMT_RF_HEADER	'MQHRFbbb'
MQFMT_RF_HEADER_2	'MQHRF2bb'
MQFMT_STRING	'MQSTRbbb'
MQFMT_TRIGGER	'MQTRIGbb'
MQFMT_WORK_INFO_HEADER	'MQHWIHbb'
MQFMT_XMIT_Q_HEADER	'MQXMITbb'

For the C programming language, the following array versions are also defined:

MQFMT_NONE_ARRAY	'b','b','b','b','b','b','b','b','b'
MQFMT_ADMIN_ARRAY	'M','Q','A','D','M','I','N','b'
MQFMT_CHANNEL_COMPLETED_ARRAY	'M','Q','C','H','C','O','M','b'
MQFMT_CICS_ARRAY	'M','Q','C','I','C','S','b','b'
MQFMT_COMMAND_1_ARRAY	'M','Q','C','M','D','1','b','b'
MQFMT_COMMAND_2_ARRAY	'M','Q','C','M','D','2','b','b'
MQFMT_DEAD_LETTER_HEADER_ARRAY	'M','Q','D','E','A','D','b','b'
MQFMT_DIST_HEADER_ARRAY	'M','Q','H','D','I','S','T','b'
MQFMT_EVENT_ARRAY	'M','Q','E','V','E','N','T','b'
MQFMT_IMS_ARRAY	'M','Q','I','M','S','b','b','b'
MQFMT_IMS_VAR_STRING_ARRAY	'M','Q','I','M','S','V','S','b'
MQFMT_MD_EXTENSION_ARRAY	'M','Q','H','M','D','E','b','b'
MQFMT_PCF_ARRAY	'M','Q','P','C','F','b','b','b'
MQFMT_REF_MSG_HEADER_ARRAY	'M','Q','H','R','E','F','b','b'
MQFMT_RF_HEADER_ARRAY	'M','Q','H','R','F','b','b','b'
MQFMT_RF_HEADER_2_ARRAY	'M','Q','H','R','F','2','b','b'
MQFMT_STRING_ARRAY	'M','Q','S','T','R','b','b','b'
MQFMT_TRIGGER_ARRAY	'M','Q','T','R','I','G','b','b'
MQFMT_WORK_INFO_HEADER_ARRAY	'M','Q','H','W','I','H','b','b'
MQFMT_XMIT_Q_HEADER_ARRAY	'M','Q','X','M','I','T','b','b'

MQGI_* (Group identifier)

See the *GroupId* field described in Chapter 10, “MQMD – Message descriptor”, on page 141.

MQ constants

MQGL_NONE X'00...00' (24 nulls)

For the C programming language, the following array version is also defined:

MQGL_NONE_ARRAY '\0','\0',...'\0','\0'

MQGMO_* (Get message options)

See the *Options* field described in Chapter 8, “MQGMO – Get-message options”, on page 95.

MQGMO_NO_WAIT	0	X'00000000'
MQGMO_NONE	0	X'00000000'
MQGMO_WAIT	1	X'00000001'
MQGMO_SYNCPOINT	2	X'00000002'
MQGMO_NO_SYNCPOINT	4	X'00000004'
MQGMO_SET_SIGNAL	8	X'00000008'
MQGMO_BROWSE_FIRST	16	X'00000010'
MQGMO_BROWSE_NEXT	32	X'00000020'
MQGMO_ACCEPT_TRUNCATED_MSG	64	X'00000040'
MQGMO_MARK_SKIP_BACKOUT	128	X'00000080'
MQGMO_MSG_UNDER_CURSOR	256	X'00000100'
MQGMO_LOCK	512	X'00000200'
MQGMO_UNLOCK	1024	X'00000400'
MQGMO_BROWSE_MSG_UNDER_CURSOR	2048	X'00000800'
MQGMO_SYNCPOINT_IF_PERSISTENT	4096	X'00001000'
MQGMO_FAIL_IF QUIESCING	8192	X'00002000'
MQGMO_CONVERT	16384	X'00004000'
MQGMO_LOGICAL_ORDER	32768	X'00008000'
MQGMO_COMPLETE_MSG	65536	X'00010000'
MQGMO_ALL_MSGS_AVAILABLE	131072	X'00020000'
MQGMO_ALL_SEGMENTS_AVAILABLE	262144	X'00040000'

MQGMO_* (Get message options structure identifier)

See the *StrucId* field described in Chapter 8, “MQGMO – Get-message options”, on page 95.

MQGMO_STRUC_ID 'GM0b'

For the C programming language, the following array version is also defined:

MQGMO_STRUC_ID_ARRAY 'G','M','0','b'

MQGMO_* (Get message options version)

See the *Version* field described in Chapter 8, “MQGMO – Get-message options”, on page 95.

MQGMO_VERSION_1	1	X'00000001'
MQGMO_VERSION_2	2	X'00000002'
MQGMO_VERSION_3	3	X'00000003'

MQGMO_CURRENT_VERSION (variable)

MQGS_* (Group status)

See the *GroupStatus* field described in Chapter 8, “MQGMO – Get-message options”, on page 95.

MQGS_NOT_IN_GROUP	'b'
MQGS_MSG_IN_GROUP	'G'
MQGS_LAST_MSG_IN_GROUP	'L'

MQHC_* (Connection handle)

See the *Hconn* parameter described in Chapter 31, “MQCONN – Connect queue manager”, on page 359 and Chapter 33, “MQDISC – Disconnect queue manager”, on page 373.

MQHC_UNUSABLE_HCONN	-1	X'FFFFFFFF'
MQHC_DEF_HCONN	0	X'00000000'

MQHO_* (Object handle)

See the *Hobj* parameter described in Chapter 29, “MQCLOSE – Close object”, on page 345.

MQHO_UNUSABLE_HOBJ	-1	X'FFFFFFFF'
MQHO_NONE	0	X'00000000'

MQIA_* (Integer attribute selector)

See the *Selectors* parameter described in Chapter 35, “MQINQ – Inquire object attributes”, on page 393 and Chapter 35, “MQINQ – Inquire object attributes”, on page 393.

MQIA_FIRST	1	X'00000001'
MQIA_APPL_TYPE	1	X'00000001'
MQIA_CODED_CHAR_SET_ID	2	X'00000002'
MQIA_CURRENT_Q_DEPTH	3	X'00000003'
MQIA_DEF_INPUT_OPEN_OPTION	4	X'00000004'
MQIA_DEF_PERSISTENCE	5	X'00000005'
MQIA_DEF_PRIORITY	6	X'00000006'
MQIA_DEFINITION_TYPE	7	X'00000007'
MQIA_HARDEN_GET_BACKOUT	8	X'00000008'
MQIA_INHIBIT_GET	9	X'00000009'
MQIA_INHIBIT_PUT	10	X'0000000A'
MQIA_MAX_HANDLES	11	X'0000000B'
MQIA_USAGE	12	X'0000000C'
MQIA_MAX_MSG_LENGTH	13	X'0000000D'
MQIA_MAX_PRIORITY	14	X'0000000E'
MQIA_MAX_Q_DEPTH	15	X'0000000F'
MQIA_MSG_DELIVERY_SEQUENCE	16	X'00000010'
MQIA_OPEN_INPUT_COUNT	17	X'00000011'
MQIA_OPEN_OUTPUT_COUNT	18	X'00000012'

MQ constants

MQIA_NAME_COUNT	19	X'00000013'
MQIA_Q_TYPE	20	X'00000014'
MQIA_RETENTION_INTERVAL	21	X'00000015'
MQIA_BACKOUT_THRESHOLD	22	X'00000016'
MQIA_SHAREABILITY	23	X'00000017'
MQIA_TRIGGER_CONTROL	24	X'00000018'
MQIA_TRIGGER_INTERVAL	25	X'00000019'
MQIA_TRIGGER_MSG_PRIORITY	26	X'0000001A'
MQIA_TRIGGER_TYPE	28	X'0000001C'
MQIA_TRIGGER_DEPTH	29	X'0000001D'
MQIA_SYNCPOINT	30	X'0000001E'
MQIA_COMMAND_LEVEL	31	X'0000001F'
MQIA_PLATFORM	32	X'00000020'
MQIA_MAX_UNCOMMITTED_MSGS	33	X'00000021'
MQIA_DIST_LISTS	34	X'00000022'
MQIA_TIME_SINCE_RESET	35	X'00000023'
MQIA_HIGH_Q_DEPTH	36	X'00000024'
MQIA_MSG_ENQ_COUNT	37	X'00000025'
MQIA_MSG_DEQ_COUNT	38	X'00000026'
MQIA_EXPIRY_INTERVAL	39	X'00000027'
MQIA_Q_DEPTH_HIGH_LIMIT	40	X'00000028'
MQIA_Q_DEPTH_LOW_LIMIT	41	X'00000029'
MQIA_Q_DEPTH_MAX_EVENT	42	X'0000002A'
MQIA_Q_DEPTH_HIGH_EVENT	43	X'0000002B'
MQIA_Q_DEPTH_LOW_EVENT	44	X'0000002C'
MQIA_SCOPE	45	X'0000002D'
MQIA_Q_SERVICE_INTERVAL_EVENT	46	X'0000002E'
MQIA_AUTHORITY_EVENT	47	X'0000002F'
MQIA_INHIBIT_EVENT	48	X'00000030'
MQIA_LOCAL_EVENT	49	X'00000031'
MQIA_REMOTE_EVENT	50	X'00000032'
MQIA_START_STOP_EVENT	52	X'00000034'
MQIA_PERFORMANCE_EVENT	53	X'00000035'
MQIA_Q_SERVICE_INTERVAL	54	X'00000036'
MQIA_CHANNEL_AUTO_DEF	55	X'00000037'
MQIA_CHANNEL_AUTO_DEF_EVENT	56	X'00000038'
MQIA_INDEX_TYPE	57	X'00000039'
MQIA_CLUSTER_WORKLOAD_LENGTH	58	X'0000003A'
MQIA_CLUSTER_Q_TYPE	59	X'0000003B'
MQIA_DEF_BIND	61	X'0000003D'
MQIA_QSG_DISP	63	X'0000003F'
MQIA_INTRA_GROUP_QUEUEING	64	X'00000040'
MQIA_IGQ_PUT_AUTHORITY	65	X'00000041'
MQIA_NAMELIST_TYPE	72	X'00000048'
MQIA_USER_LIST	2000	X'000007D0'
MQIA_LAST	2000	X'000007D0'
MQIA_LAST_USED	(variable)	

MQIAUT_* (IMS authenticator)

See the *Authenticator* field described in Chapter 9, "MQIIH – IMS information header", on page 133.

MQIAUT_NONE 'bbbbbbbb'

For the C programming language, the following array version is also defined:

```
MQIAUT_NONE_ARRAY          'b','b','b','b','b','b','b','b'
```

MQIAV_* (Integer attribute value)

See the *IntAttrs* parameter described in Chapter 35, “MQINQ – Inquire object attributes”, on page 393.

```
MQIAV_UNDEFINED           -2          X'FFFFFFFE'
MQIAV_NOT_APPLICABLE     -1          X'FFFFFFF'
```

MQICM_* (IMS commit mode)

See the *CommitMode* field described in Chapter 9, “MQIIH – IMS information header”, on page 133.

```
MQICM_COMMIT_THEN_SEND   '0'
MQICM_SEND_THEN_COMMIT  '1'
```

MQIGQ_* (Intra-group queuing)

See the *IntraGroupQueuing* attribute described in Chapter 43, “Attributes for the queue manager”, on page 501.

```
MQIGQ_DISABLED           0          X'00000000'
MQIGQ_ENABLED            1          X'00000001'
```

MQIGQPA_* (Intra-group queuing put authority)

See the *IGQPutAuthority* attribute described in Chapter 43, “Attributes for the queue manager”, on page 501.

```
MQIGQPA_DEFAULT         1          X'00000001'
MQIGQPA_CONTEXT         2          X'00000002'
MQIGQPA_ONLY_IGQ       3          X'00000003'
MQIGQPA_ALTERNATE_OR_IGQ 4          X'00000004'
```

MQIIH_* (IMS header flags)

See the *Flags* field described in Chapter 9, “MQIIH – IMS information header”, on page 133.

```
| MQIIH_NONE              0          X'00000000'
| MQIIH_UNLIMITED_EXPIRATION 0          X'00000000'
| MQIIH_PASS_EXPIRATION   1          X'00000001'
| MQIIH_REPLY_FORMAT_NONE 8          X'00000008'
```

MQ constants

MQIIH_* (IMS header length)

See the *StrucLength* field described in Chapter 9, “MQIIH – IMS information header”, on page 133.

MQIIH_LENGTH_1	84	X'00000054'
----------------	----	-------------

MQIIH_* (IMS header structure identifier)

See the *StrucId* field described in Chapter 9, “MQIIH – IMS information header”, on page 133.

MQIIH_STRUC_ID	'IIHb'
----------------	--------

For the C programming language, the following array version is also defined:

MQIIH_STRUC_ID_ARRAY	'I','I','H','b'
----------------------	-----------------

MQIIH_* (IMS header version)

See the *Version* field described in Chapter 9, “MQIIH – IMS information header”, on page 133.

MQIIH_VERSION_1	1	X'00000001'
MQIIH_CURRENT_VERSION	1	X'00000001'

MQISS_* (IMS security scope)

See the *SecurityScope* field described in Chapter 9, “MQIIH – IMS information header”, on page 133.

MQISS_CHECK	'C'
MQISS_FULL	'F'

MQIT_* (Index type)

See the *IndexType* attribute described in Chapter 40, “Attributes for queues”, on page 457.

MQIT_NONE	0	X'00000000'
MQIT_MSG_ID	1	X'00000001'
MQIT_CORREL_ID	2	X'00000002'
MQIT_MSG_TOKEN	4	X'00000004'
MQIT_GROUP_ID	5	X'00000005'

MQITIL_* (IMS transaction instance identifier)

See the *TranInstanceId* field described in Chapter 9, “MQIIH – IMS information header”, on page 133.

MQITIL_NONE X'00...00' (16 nulls)

For the C programming language, the following array version is also defined:

MQITIL_NONE_ARRAY '\0', '\0', ... '\0', '\0'

MQITS_* (IMS transaction state)

See the *TranState* field described in Chapter 9, “MQIIH – IMS information header”, on page 133.

MQITS_IN_CONVERSATION 'C'
 MQITS_NOT_IN_CONVERSATION 'b'
 MQITS_ARCHITECTED 'A'

MQMD_* (Message descriptor structure identifier)

See the *StrucId* field described in Chapter 10, “MQMD – Message descriptor”, on page 141.

MQMD_STRUC_ID 'MDbb'

For the C programming language, the following array version is also defined:

MQMD_STRUC_ID_ARRAY 'M', 'D', 'b', 'b'

MQMD_* (Message descriptor version)

See the *Version* field described in Chapter 10, “MQMD – Message descriptor”, on page 141.

MQMD_VERSION_1	1	X'00000001'
MQMD_VERSION_2	2	X'00000002'
MQMD_CURRENT_VERSION	(variable)	

MQMDE_* (Message descriptor extension length)

See the *StrucLength* field described in Chapter 11, “MQMDE – Message descriptor extension”, on page 203.

MQMDE_LENGTH_2 72 X'00000048'

MQMF_* (Message-flags masks)

See Appendix E, “Report options and message flags”, on page 575.

MQMF_ACCEPT_UNSUP_MASK	-1048576	X'FFF0000'
MQMF_REJECT_UNSUP_MASK	4095	X'0000FFF'
MQMF_ACCEPT_UNSUP_IF_XMIT_MASK	1044480	X'000FF00'

MQMI_* (Message identifier)

See the *MsgId* field described in Chapter 10, “MQMD – Message descriptor”, on page 141.

MQMI_NONE X'00...00' (24 nulls)

For the C programming language, the following array version is also defined:

MQMI_NONE_ARRAY '\0','\0',...'\0','\0'

MQMO_* (Match options)

See the *MatchOptions* field described in Chapter 8, “MQGMO – Get-message options”, on page 95.

MQMO_NONE	0	X'0000000'
MQMO_MATCH_MSG_ID	1	X'0000001'
MQMO_MATCH_CORREL_ID	2	X'0000002'
MQMO_MATCH_GROUP_ID	4	X'0000004'
MQMO_MATCH_MSG_SEQ_NUMBER	8	X'0000008'
MQMO_MATCH_OFFSET	16	X'0000010'
MQMO_MATCH_MSG_TOKEN	32	X'0000020'

MQMT_* (Message type)

See the *MsgType* field described in Chapter 10, “MQMD – Message descriptor”, on page 141.

MQMT_SYSTEM_FIRST	1	X'0000001'
MQMT_REQUEST	1	X'0000001'
MQMT_REPLY	2	X'0000002'
MQMT_REPORT	4	X'0000004'
MQMT_DATAGRAM	8	X'0000008'
MQMT_MQE_FIELDS_FROM_MQE	112	X'0000070'
MQMT_MQE_FIELDS	113	X'0000071'
MQMT_SYSTEM_LAST	65535	X'000FFFF'
MQMT_APPL_FIRST	65536	X'0001000'
MQMT_APPL_LAST	99999999	X'3B9AC9FF'

MQ constants

MQOT_* (Object type)

See the *ObjectType* field described in Chapter 12, “MQOD – Object descriptor”, on page 211.

MQOT_Q	1	X'00000001'
MQOT_NAMELIST	2	X'00000002'
MQOT_PROCESS	3	X'00000003'
MQOT_Q_MGR	5	X'00000005'

MQPER_* (Persistence)

See the *Persistence* field described in Chapter 10, “MQMD – Message descriptor”, on page 141, and the *DefPersistence* attribute described in Chapter 40, “Attributes for queues”, on page 457.

MQPER_NOT_PERSISTENT	0	X'00000000'
MQPER_PERSISTENT	1	X'00000001'
MQPER_PERSISTENCE_AS_Q_DEF	2	X'00000002'

MQPL_* (Platform)

See the *Platform* attribute described in Chapter 43, “Attributes for the queue manager”, on page 501.

MQPL_MVS	1	X'00000001'
MQPL_OS390	1	X'00000001'
MQPL_ZOS	1	X'00000001'
MQPL_OS2	2	X'00000002'
MQPL_AIX	3	X'00000003'
MQPL_UNIX	3	X'00000003'
MQPL_OS400	4	X'00000004'
MQPL_WINDOWS	5	X'00000005'
MQPL_WINDOWS_NT	11	X'0000000B'
MQPL_VMS	12	X'0000000C'
MQPL_NSK	13	X'0000000D'

MQPMO_* (Put message options)

See the *Options* field described in Chapter 14, “MQPMO – Put-message options”, on page 229.

MQPMO_NONE	0	X'00000000'
MQPMO_SYNCPOINT	2	X'00000002'
MQPMO_NO_SYNCPOINT	4	X'00000004'
MQPMO_DEFAULT_CONTEXT	32	X'00000020'
MQPMO_NEW_MSG_ID	64	X'00000040'
MQPMO_NEW_CORREL_ID	128	X'00000080'
MQPMO_PASS_IDENTITY_CONTEXT	256	X'00000100'
MQPMO_PASS_ALL_CONTEXT	512	X'00000200'
MQPMO_SET_IDENTITY_CONTEXT	1024	X'00000400'
MQPMO_SET_ALL_CONTEXT	2048	X'00000800'

MQPMO_ALTERNATE_USER_AUTHORITY	4096	X'00001000'
MQPMO_FAIL_IF QUIESCING	8192	X'00002000'
MQPMO_NO_CONTEXT	16384	X'00004000'
MQPMO_LOGICAL_ORDER	32768	X'00008000'

MQPMO_* (Put message options structure length)

See Chapter 14, “MQPMO – Put-message options”, on page 229.

MQPMO_CURRENT_LENGTH	(variable)
----------------------	------------

MQPMO_* (Put message options structure identifier)

See the *StrucId* field described in Chapter 14, “MQPMO – Put-message options”, on page 229.

MQPMO_STRUC_ID	'PM0b'
----------------	--------

For the C programming language, the following array version is also defined:

MQPMO_STRUC_ID_ARRAY	'P','M','0','b'
----------------------	-----------------

MQPMO_* (Put message options version)

See the *Version* field described in Chapter 14, “MQPMO – Put-message options”, on page 229.

MQPMO_VERSION_1	1	X'00000001'
MQPMO_VERSION_2	2	X'00000002'
MQPMO_CURRENT_VERSION	(variable)	

MQPMRF_* (Put message record field flags)

See the *PutMsgRecFields* field described in Chapter 6, “MQDH – Distribution header”, on page 75.

MQPMRF_NONE	0	X'00000000'
MQPMRF_MSG_ID	1	X'00000001'
MQPMRF_CORREL_ID	2	X'00000002'
MQPMRF_GROUP_ID	4	X'00000004'
MQPMRF_FEEDBACK	8	X'00000008'
MQPMRF_ACCOUNTING_TOKEN	16	X'00000010'

MQPRI_* (Priority)

See the *Priority* field described in Chapter 10, “MQMD – Message descriptor”, on page 141.

MQ constants

MQPRI_PRIORITY_AS_Q_DEF	-1	X'FFFFFFFF'
-------------------------	----	-------------

MQQA_* (Inhibit get)

See the *InhibitGet* attribute described in Chapter 40, "Attributes for queues", on page 457.

MQQA_GET_ALLOWED	0	X'00000000'
MQQA_GET_INHIBITED	1	X'00000001'

MQQA_* (Inhibit put)

See the *InhibitPut* attribute described in Chapter 40, "Attributes for queues", on page 457.

MQQA_PUT_ALLOWED	0	X'00000000'
MQQA_PUT_INHIBITED	1	X'00000001'

MQQA_* (Backout hardening)

See the *HardenGetBackout* attribute described in Chapter 40, "Attributes for queues", on page 457.

MQQA_BACKOUT_NOT_HARDENED	0	X'00000000'
MQQA_BACKOUT_HARDENED	1	X'00000001'

MQQA_* (Queue shareability)

See the *Shareability* attribute described in Chapter 40, "Attributes for queues", on page 457.

MQQA_NOT_SHAREABLE	0	X'00000000'
MQQA_SHAREABLE	1	X'00000001'

MQQDT_* (Queue definition type)

See the *DefinitionType* attribute described in Chapter 40, "Attributes for queues", on page 457.

MQQDT_PREDEFINED	1	X'00000001'
MQQDT_PERMANENT_DYNAMIC	2	X'00000002'
MQQDT_TEMPORARY_DYNAMIC	3	X'00000003'
MQQDT_SHARED_DYNAMIC	4	X'00000004'

MQQSGD_* (Queue-sharing group disposition)

See the *QSGDisp* attribute described in Chapter 40, "Attributes for queues", on page 457.

MQQSGD_Q_MGR	0	X'00000000'
MQQSGD_COPY	1	X'00000001'

MQQSGD_SHARED	2	X'00000002'
---------------	---	-------------

MQQSIE_* (Service interval events)

See the *QServiceIntervalEvent* attribute described in Chapter 40, "Attributes for queues", on page 457.

MQQSIE_NONE	0	X'00000000'
MQQSIE_HIGH	1	X'00000001'
MQQSIE_OK	2	X'00000002'

MQQT_* (Queue type)

See the *QType* attribute described in Chapter 40, "Attributes for queues", on page 457.

MQQT_LOCAL	1	X'00000001'
MQQT_MODEL	2	X'00000002'
MQQT_ALIAS	3	X'00000003'
MQQT_REMOTE	6	X'00000006'
MQQT_CLUSTER	7	X'00000007'

MQRC_* (Reason code)

See Appendix A, "Return codes", on page 527, and the *Feedback* field described in Chapter 10, "MQMD – Message descriptor", on page 141. For complete lists of reason codes, in both alphabetic and numeric order, see:

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

MQRFH_* (Rules and formatting header flags)

See the *Flags* field described in Chapter 16, "MQRFH – Rules and formatting header", on page 255.

MQRFH_NONE	0	X'00000000'
------------	---	-------------

MQRFH_* (Rules and formatting header length)

See the *StrucLength* field described in Chapter 16, "MQRFH – Rules and formatting header", on page 255.

MQRFH_STRUC_LENGTH_FIXED	32	X'00000020'
MQRFH_STRUC_LENGTH_FIXED_2	36	X'00000024'

MQRFH_* (Rules and formatting header structure identifier)

See the *StrucId* field described in Chapter 16, "MQRFH – Rules and formatting header", on page 255.

MQ constants

MQRMHF_LAST	1	X'00000001'
-------------	---	-------------

MQRO_* (Report options)

See the *Report* field described in Chapter 10, “MQMD – Message descriptor”, on page 141.

MQRO_NEW_MSG_ID	0	X'00000000'
MQRO_COPY_MSG_ID_TO_CORREL_ID	0	X'00000000'
MQRO_DEAD_LETTER_Q	0	X'00000000'
MQRO_NONE	0	X'00000000'
MQRO_PAN	1	X'00000001'
MQRO_NAN	2	X'00000002'
MQRO_PASS_CORREL_ID	64	X'00000040'
MQRO_PASS_MSG_ID	128	X'00000080'
MQRO_COA	256	X'00000100'
MQRO_COA_WITH_DATA	768	X'00000300'
MQRO_COA_WITH_FULL_DATA	1792	X'00000700'
MQRO_COD	2048	X'00000800'
MQRO_COD_WITH_DATA	6144	X'00001800'
MQRO_COD_WITH_FULL_DATA	14336	X'00003800'
MQRO_EXPIRATION	2097152	X'00200000'
MQRO_EXPIRATION_WITH_DATA	6291456	X'00600000'
MQRO_EXPIRATION_WITH_FULL_DATA	14680064	X'00E00000'
MQRO_EXCEPTION	16777216	X'01000000'
MQRO_EXCEPTION_WITH_DATA	50331648	X'03000000'
MQRO_EXCEPTION_WITH_FULL_DATA	117440512	X'07000000'
MQRO_DISCARD_MSG	134217728	X'08000000'

MQRO_* (Report-options masks)

See Appendix E, “Report options and message flags”, on page 575.

MQRO_REJECT_UNSUP_MASK	270270464	X'101C0000'
MQRO_ACCEPT_UNSUP_MASK	-270532353	X'EFE000FF'
MQRO_ACCEPT_UNSUP_IF_XMIT_MASK	261888	X'0003FF00'

MQSCO_* (Queue scope)

See the *Scope* attribute described in Chapter 40, “Attributes for queues”, on page 457.

MQSCO_Q_MGR	1	X'00000001'
MQSCO_CELL	2	X'00000002'

MQSCO_* (SSL configuration options structure identifier)

See the *StrucId* field described in Chapter 20, “MQSCO – SSL configuration options”, on page 285.

MQSCO_STRUC_ID	'SC0b'
----------------	--------

MQ constants

For the C programming language, the following array version is also defined:

```
MQSCO_STRUC_ID_ARRAY          'S','C','0','b'
```

MQSCO_* (SSL configuration options version)

See the *Version* field described in Chapter 20, “MQSCO – SSL configuration options”, on page 285.

```
MQSCO_VERSION_1              1          X'00000001'  
MQSCO_CURRENT_VERSION        (variable)
```

MQSEG_* (Segmentation)

See the *Segmentation* field described in Chapter 8, “MQGMO – Get-message options”, on page 95.

```
MQSEG_INHIBITED              'b'  
MQSEG_ALLOWED                'A'
```

MQSID_* (Security identifier)

See the *AlternateSecurityId* field described in Chapter 12, “MQOD – Object descriptor”, on page 211.

```
MQSID_NONE                    X'00...00' (40 nulls)
```

For the C programming language, the following array version is also defined:

```
MQSID_NONE_ARRAY             '\0','\0',...'\0','\0'
```

MQSIDT_* (Security identifier type)

See the *AlternateSecurityId* field described in Chapter 12, “MQOD – Object descriptor”, on page 211.

```
MQSIDT_NONE                   X'00'  
MQSIDT_NT_SECURITY_ID         X'01'
```

MQSP_* (Syncpoint)

See the *SyncPoint* attribute described in Chapter 43, “Attributes for the queue manager”, on page 501.

```
MQSP_NOT_AVAILABLE          0          X'00000000'  
MQSP_AVAILABLE              1          X'00000001'
```

MQSS_* (Segment status)

See the *SegmentStatus* field described in Chapter 8, “MQGMO – Get-message options”, on page 95.

MQSS_NOT_A_SEGMENT	'b'
MQSS_LAST_SEGMENT	'L'
MQSS_SEGMENT	'S'

MQTC_* (Trigger control)

See the *TriggerControl* attribute described in Chapter 40, “Attributes for queues”, on page 457.

MQTC_OFF	0	X'00000000'
MQTC_ON	1	X'00000001'

MQTM_* (Trigger message structure identifier)

See the *StrucId* field described in Chapter 21, “MQTM – Trigger message”, on page 291.

MQTM_STRUC_ID	'TMbb'
---------------	--------

For the C programming language, the following array version is also defined:

MQTM_STRUC_ID_ARRAY	'T','M','b','b'
---------------------	-----------------

MQTM_* (Trigger message structure version)

See the *Version* field described in Chapter 21, “MQTM – Trigger message”, on page 291.

MQTM_VERSION_1	1	X'00000001'
MQTM_CURRENT_VERSION	1	X'00000001'

MQTMC_* (Trigger message character format structure identifier)

See the *StrucId* field described in Chapter 22, “MQTMC2 – Trigger message 2 (character format)”, on page 299.

MQTMC_STRUC_ID	'TMCb'
----------------	--------

For the C programming language, the following array version is also defined:

MQTMC_STRUC_ID_ARRAY	'T','M','C','b'
----------------------	-----------------

MQ constants

MQTMC_* (Trigger message character format structure version)

See the *Version* field described in Chapter 22, “MQTMC2 – Trigger message 2 (character format)”, on page 299.

MQTMC_VERSION_1	'bbb1'
MQTMC_VERSION_2	'bbb2'
MQTMC_CURRENT_VERSION	'bbb2'

For the C programming language, the following array versions are also defined:

MQTMC_VERSION_1_ARRAY	'b','b','b','1'
MQTMC_VERSION_2_ARRAY	'b','b','b','2'
MQTMC_CURRENT_VERSION_ARRAY	'b','b','b','2'

MQTT_* (Trigger type)

See the *TriggerType* attribute described in Chapter 40, “Attributes for queues”, on page 457.

MQTT_NONE	0	X'00000000'
MQTT_FIRST	1	X'00000001'
MQTT_EVERY	2	X'00000002'
MQTT_DEPTH	3	X'00000003'

MQUS_* (Usage)

See the *Usage* attribute described in Chapter 40, “Attributes for queues”, on page 457.

MQUS_NORMAL	0	X'00000000'
MQUS_TRANSMISSION	1	X'00000001'

MQWI_* (Wait interval)

See the *WaitInterval* field described in Chapter 8, “MQGMO – Get-message options”, on page 95.

MQWI_UNLIMITED	-1	X'FFFFFFFF'
----------------	----	-------------

MQWIH_* (Workload information header flags)

See the *Flags* field described in Chapter 23, “MQWIH – Work information header”, on page 305.

MQWIH_NONE	0	X'00000000'
------------	---	-------------

MQWIH_* (Workload information header structure length)

See the *StrucLength* field described in Chapter 23, “MQWIH – Work information header”, on page 305.

MQWIH_LENGTH_1	120	X'00000078'
MQWIH_CURRENT_LENGTH	120	X'00000078'

MQWIH_* (Workload information header structure identifier)

See the *StrucId* field described in Chapter 23, “MQWIH – Work information header”, on page 305.

MQWIH_STRUC_ID	'WIHb'
----------------	--------

For the C programming language, the following array version is also defined:

MQWIH_STRUC_ID_ARRAY	'W','I','H','b'
----------------------	-----------------

MQWIH_* (Workload information header version)

See the *Version* field described in Chapter 23, “MQWIH – Work information header”, on page 305.

MQWIH_VERSION_1	1	X'00000001'
MQWIH_CURRENT_VERSION	1	X'00000001'

MQXC_* (Exit command identifier)

See the *ExitCommand* field described in Chapter 24, “MQXP – Exit parameter block”, on page 311.

MQXC_MQOPEN	1	X'00000001'
MQXC_MQCLOSE	2	X'00000002'
MQXC_MQGET	3	X'00000003'
MQXC_MQPUT	4	X'00000004'
MQXC_MQPUT1	5	X'00000005'
MQXC_MQINQ	6	X'00000006'
MQXC_MQSET	8	X'00000008'
MQXC_MQBACK	9	X'00000009'
MQXC_MQCMIT	10	X'0000000A'

MQXCC_* (Exit response)

See the *ExitResponse* field described in Chapter 24, “MQXP – Exit parameter block”, on page 311.

MQXCC_SKIP_FUNCTION	-2	X'FFFFFFF'
---------------------	----	------------

MQXR_* (Exit reason)

See the *ExitReason* field described in Chapter 24, “MQXP – Exit parameter block”, on page 311.

MQXR_BEFORE	1	X'00000001'
MQXR_AFTER	2	X'00000002'

MQXT_* (Exit identifier)

See the *ExitId* field described in Chapter 24, “MQXP – Exit parameter block”, on page 311.

MQXT_API_CROSSING_EXIT	1	X'00000001'
------------------------	---	-------------

MQXUA_* (Exit user area)

See the *ExitUserArea* field described in Chapter 24, “MQXP – Exit parameter block”, on page 311.

MQXUA_NONE X'00...00' (16 nulls)

For the C programming language, the following array version is also defined:

MQXUA_NONE_ARRAY '\0','\0',...'\0','\0'

MQ constants

Appendix C. Rules for validating MQI options

This appendix lists the situations that produce an MQRC_OPTIONS_ERROR reason code from an MQCONN, MQOPEN, MQPUT, MQPUT1, MQGET, or MQCLOSE call.

MQOPEN call

For the options of the MQOPEN call:

- At least *one* of the following must be specified:
 - MQOO_BROWSE
 - MQOO_INPUT_AS_Q_DEF
 - MQOO_INPUT_EXCLUSIVE
 - MQOO_INPUT_SHARED
 - MQOO_INQUIRE
 - MQOO_OUTPUT
 - MQOO_SET
- Only *one* of the following is allowed:
 - MQOO_INPUT_AS_Q_DEF
 - MQOO_INPUT_EXCLUSIVE
 - MQOO_INPUT_SHARED
- Only *one* of the following is allowed:
 - MQOO_BIND_ON_OPEN
 - MQOO_BIND_NOT_FIXED
 - MQOO_BIND_AS_Q_DEF

Note: The options listed above are mutually exclusive. However, as the value of MQOO_BIND_AS_Q_DEF is zero, specifying it with either of the other two bind options does not result in reason code MQRC_OPTIONS_ERROR. MQOO_BIND_AS_Q_DEF is provided to aid program documentation.

- If MQOO_SAVE_ALL_CONTEXT is specified, one of the MQOO_INPUT_* options must also be specified.
- If one of the MQOO_SET_*_CONTEXT or MQOO_PASS_*_CONTEXT options is specified, MQOO_OUTPUT must also be specified.

MQPUT call

For the put-message options:

- The combination of MQPMO_SYNCPOINT and MQPMO_NO_SYNCPOINT is not allowed.
- Only *one* of the following is allowed:
 - MQPMO_DEFAULT_CONTEXT
 - MQPMO_NO_CONTEXT
 - MQPMO_PASS_ALL_CONTEXT
 - MQPMO_PASS_IDENTITY_CONTEXT
 - MQPMO_SET_ALL_CONTEXT
 - MQPMO_SET_IDENTITY_CONTEXT
- MQPMO_ALTERNATE_USER_AUTHORITY is not allowed (it is valid only on the MQPUT1 call).

MQPUT1 call

MQPUT1 call

For the put-message options, the rules are the same as for the MQPUT call, except for the following:

- MQPMO_ALTERNATE_USER_AUTHORITY is allowed.
 - MQPMO_LOGICAL_ORDER is *not* allowed.
-

MQGET call

For the get-message options:

- Only *one* of the following is allowed:
 - MQGMO_NO_SYNCPOINT
 - MQGMO_SYNCPOINT
 - MQGMO_SYNCPOINT_IF_PERSISTENT
 - Only *one* of the following is allowed:
 - MQGMO_BROWSE_FIRST
 - MQGMO_BROWSE_MSG_UNDER_CURSOR
 - MQGMO_BROWSE_NEXT
 - MQGMO_MSG_UNDER_CURSOR
 - MQGMO_SYNCPOINT is not allowed with any of the following:
 - MQGMO_BROWSE_FIRST
 - MQGMO_BROWSE_MSG_UNDER_CURSOR
 - MQGMO_BROWSE_NEXT
 - MQGMO_LOCK
 - MQGMO_UNLOCK
 - MQGMO_SYNCPOINT_IF_PERSISTENT is not allowed with any of the following:
 - MQGMO_BROWSE_FIRST
 - MQGMO_BROWSE_MSG_UNDER_CURSOR
 - MQGMO_BROWSE_NEXT
 - MQGMO_COMPLETE_MSG
 - MQGMO_UNLOCK
 - MQGMO_MARK_SKIP_BACKOUT requires MQGMO_SYNCPOINT to be specified.
 - The combination of MQGMO_WAIT and MQGMO_SET_SIGNAL is not allowed.
 - If MQGMO_LOCK is specified, one of the following must also be specified:
 - MQGMO_BROWSE_FIRST
 - MQGMO_BROWSE_MSG_UNDER_CURSOR
 - MQGMO_BROWSE_NEXT
 - If MQGMO_UNLOCK is specified, only the following are allowed:
 - MQGMO_NO_SYNCPOINT
 - MQGMO_NO_WAIT
-

MQCLOSE call

For the options of the MQCLOSE call:

- The combination of MQCO_DELETE and MQCO_DELETE_PURGE is not allowed.

Appendix D. Machine encodings

This appendix describes the structure of the *Encoding* field in the message descriptor (see Chapter 10, “MQMD – Message descriptor”, on page 141).

The *Encoding* field is a 32-bit integer that is divided into four separate subfields; these subfields identify:

- The encoding used for binary integers
- The encoding used for packed-decimal integers
- The encoding used for floating-point numbers
- Reserved bits

Each subfield is identified by a bit mask which has 1-bits in the positions corresponding to the subfield, and 0-bits elsewhere. The bits are numbered such that bit 0 is the most significant bit, and bit 31 the least significant bit. The following masks are defined:

MQENC_INTEGER_MASK

Mask for binary-integer encoding.

This subfield occupies bit positions 28 through 31 within the *Encoding* field.

MQENC_DECIMAL_MASK

Mask for packed-decimal-integer encoding.

This subfield occupies bit positions 24 through 27 within the *Encoding* field.

MQENC_FLOAT_MASK

Mask for floating-point encoding.

This subfield occupies bit positions 20 through 23 within the *Encoding* field.

MQENC_RESERVED_MASK

Mask for reserved bits.

This subfield occupies bit positions 0 through 19 within the *Encoding* field.

Binary-integer encoding

The following values are valid for the binary-integer encoding:

MQENC_INTEGER_UNDEFINED

Undefined integer encoding.

Binary integers are represented using an encoding that is undefined.

MQENC_INTEGER_NORMAL

Normal integer encoding.

Binary integers are represented in the conventional way:

- The least significant byte in the number has the highest address of any of the bytes in the number; the most significant byte has the lowest address

Machine encodings

- The least significant bit in each byte is adjacent to the byte with the next higher address; the most significant bit in each byte is adjacent to the byte with the next lower address

MQENC_INTEGER_REVERSED

Reversed integer encoding.

Binary integers are represented in the same way as `MQENC_INTEGER_NORMAL`, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as `MQENC_INTEGER_NORMAL`.

Packed-decimal-integer encoding

The following values are valid for the packed-decimal-integer encoding:

MQENC_DECIMAL_UNDEFINED

Undefined packed-decimal encoding.

Packed-decimal integers are represented using an encoding that is undefined.

MQENC_DECIMAL_NORMAL

Normal packed-decimal encoding.

Packed-decimal integers are represented in the conventional way:

- Each decimal digit in the printable form of the number is represented in packed decimal by a single hexadecimal digit in the range `X'0'` through `X'9'`. Each hexadecimal digit occupies four bits, and so each byte in the packed decimal number represents two decimal digits in the printable form of the number.
- The least significant byte in the packed-decimal number is the byte which contains the least significant decimal digit. Within that byte, the most significant four bits contain the least significant decimal digit, and the least significant four bits contain the sign. The sign is either `X'C'` (positive), `X'D'` (negative), or `X'F'` (unsigned).
- The least significant byte in the number has the highest address of any of the bytes in the number; the most significant byte has the lowest address.
- The least significant bit in each byte is adjacent to the byte with the next higher address; the most significant bit in each byte is adjacent to the byte with the next lower address.

MQENC_DECIMAL_REVERSED

Reversed packed-decimal encoding.

Packed-decimal integers are represented in the same way as `MQENC_DECIMAL_NORMAL`, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as `MQENC_DECIMAL_NORMAL`.

Floating-point encoding

The following values are valid for the floating-point encoding:

MQENC_FLOAT_UNDEFINED

Undefined floating-point encoding.

Floating-point numbers are represented using an encoding that is undefined.

MQENC_FLOAT_IEEE_NORMAL

Normal IEEE float encoding.

Floating-point numbers are represented using the standard IEEE⁵ floating-point format, with the bytes arranged as follows:

- The least significant byte in the mantissa has the highest address of any of the bytes in the number; the byte containing the exponent has the lowest address
- The least significant bit in each byte is adjacent to the byte with the next higher address; the most significant bit in each byte is adjacent to the byte with the next lower address

Details of the IEEE float encoding may be found in IEEE Standard 754.

MQENC_FLOAT_IEEE_REVERSED

Reversed IEEE float encoding.

Floating-point numbers are represented in the same way as MQENC_FLOAT_IEEE_NORMAL, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as MQENC_FLOAT_IEEE_NORMAL.

MQENC_FLOAT_S390

System/390 architecture float encoding.

Floating-point numbers are represented using the standard System/390 floating-point format; this is also used by System/370[®].

Constructing encodings

To construct a value for the *Encoding* field in MQMD, the relevant constants that describe the required encodings can be:

- Added together, or
- Combined using the bitwise OR operation (if the programming language supports bit operations)

Whichever method is used, combine only one of the MQENC_INTEGER_* encodings with one of the MQENC_DECIMAL_* encodings and one of the MQENC_FLOAT_* encodings.

Analyzing encodings

The *Encoding* field contains subfields; because of this, applications that need to examine the integer, packed decimal, or float encoding should use one of the techniques described below.

Using bit operations

If the programming language supports bit operations, the following steps should be performed:

1. Select one of the following values, according to the type of encoding required:
 - MQENC_INTEGER_MASK for the binary integer encoding
 - MQENC_DECIMAL_MASK for the packed decimal integer encoding
 - MQENC_FLOAT_MASK for the floating point encoding

Call the value A.

5. The Institute of Electrical and Electronics Engineers

Machine encodings

- Combine the *Encoding* field with *A* using the bitwise AND operation; call the result *B*.
- B* is the encoding required, and can be tested for equality with each of the values that is valid for that type of encoding.

Using arithmetic

If the programming language *does not* support bit operations, the following steps should be performed using integer arithmetic:

- Select one of the following values, according to the type of encoding required:
 - 1 for the binary integer encoding
 - 16 for the packed decimal integer encoding
 - 256 for the floating point encoding

Call the value *A*.

- Divide the value of the *Encoding* field by *A*; call the result *B*.
- Divide *B* by 16; call the result *C*.
- Multiply *C* by 16 and subtract from *B*; call the result *D*.
- Multiply *D* by *A*; call the result *E*.
- E* is the encoding required, and can be tested for equality with each of the values that is valid for that type of encoding.

Summary of machine architecture encodings

Encodings for machine architectures are shown in Table 99.

Table 99. Summary of encodings for machine architectures

Machine architecture	Binary integer encoding	Packed-decimal integer encoding	Floating-point encoding
OS/400	normal	normal	IEEE normal
Intel® x86	reversed	reversed	IEEE reversed
PowerPC	normal	normal	IEEE normal
System/390	normal	normal	System/390

Appendix E. Report options and message flags

This appendix concerns the *Report* and *MsgFlags* fields that are part of the message descriptor MQMD specified on the MQGET, MQPUT, and MQPUT1 calls (see Chapter 10, “MQMD – Message descriptor”, on page 141). The appendix describes:

- The structure of the report field and how the queue manager processes it
- How an application should analyze the report field
- The structure of the message-flags field

Structure of the report field

The *Report* field is a 32-bit integer that is divided into three separate subfields. These subfields identify:

- Report options that are rejected if the local queue manager does not recognize them
- Report options that are always accepted, even if the local queue manager does not recognize them
- Report options that are accepted only if certain other conditions are satisfied

Each subfield is identified by a bit mask which has 1-bits in the positions corresponding to the subfield, and 0-bits elsewhere. Note that the bits in a subfield are not necessarily adjacent. The bits are numbered such that bit 0 is the most significant bit, and bit 31 the least significant bit. The following masks are defined to identify the subfields:

MQRO_REJECT_UNSUP_MASK

Mask for unsupported report options that are rejected.

This mask identifies the bit positions within the *Report* field where report options which are not supported by the local queue manager will cause the MQPUT or MQPUT1 call to fail with completion code MQCC_FAILED and reason code MQRC_REPORT_OPTIONS_ERROR.

This subfield occupies bit positions 3, and 11 through 13.

MQRO_ACCEPT_UNSUP_MASK

Mask for unsupported report options that are accepted.

This mask identifies the bit positions within the *Report* field where report options which are not supported by the local queue manager will nevertheless be accepted on the MQPUT or MQPUT1 calls. Completion code MQCC_WARNING with reason code MQRC_UNKNOWN_REPORT_OPTION are returned in this case.

This subfield occupies bit positions 0 through 2, 4 through 10, and 24 through 31.

The following report options are included in this subfield:

MQRO_COPY_MSG_ID_TO_CORREL_ID
MQRO_DEAD_LETTER_Q
MQRO_DISCARD_MSG
MQRO_EXCEPTION
MQRO_EXCEPTION_WITH_DATA
MQRO_EXCEPTION_WITH_FULL_DATA
MQRO_EXPIRATION

Report options

MQRO_EXPIRATION_WITH_DATA
MQRO_EXPIRATION_WITH_FULL_DATA
MQRO_NAN
MQRO_NEW_MSG_ID
MQRO_NONE
MQRO_PAN
MQRO_PASS_CORREL_ID
MQRO_PASS_MSG_ID

MQRO_ACCEPT_UNSUP_IF_XMIT_MASK

Mask for unsupported report options that are accepted only in certain circumstances.

This mask identifies the bit positions within the *Report* field where report options which are not supported by the local queue manager will nevertheless be accepted on the MQPUT or MQPUT1 calls *provided* that both of the following conditions are satisfied:

- The message is destined for a remote queue manager.
- The application is not putting the message directly on a local transmission queue (that is, the queue identified by the *ObjectQMGrName* and *ObjectName* fields in the object descriptor specified on the MQOPEN or MQPUT1 call is not a local transmission queue).

Completion code MQCC_WARNING with reason code MQRC_UNKNOWN_REPORT_OPTION are returned if these conditions are satisfied, and MQCC_FAILED with reason code MQRC_REPORT_OPTIONS_ERROR if not.

This subfield occupies bit positions 14 through 23.

The following report options are included in this subfield:

MQRO_COA
MQRO_COA_WITH_DATA
MQRO_COA_WITH_FULL_DATA
MQRO_COD
MQRO_COD_WITH_DATA
MQRO_COD_WITH_FULL_DATA

If there are any options specified in the *Report* field which the queue manager does not recognize, the queue manager checks each subfield in turn by using the bitwise AND operation to combine the *Report* field with the mask for that subfield. If the result of that operation is not zero, the completion code and reason codes described above are returned.

If MQCC_WARNING is returned, it is not defined which reason code is returned if other warning conditions exist.

The ability to specify and have accepted report options which are not recognized by the local queue manager is useful when it is desired to send a message with a report option which will be recognized and processed by a *remote* queue manager.

Analyzing the report field

The *Report* field contains subfields; because of this, applications that need to check whether the sender of the message requested a particular report should use one of the techniques described below.

Using bit operations

If the programming language supports bit operations, the following steps should be performed:

1. Select one of the following values, according to the type of report to be checked:
 - MQRO_COA_WITH_FULL_DATA for COA report
 - MQRO_COD_WITH_FULL_DATA for COD report
 - MQRO_EXCEPTION_WITH_FULL_DATA for exception report
 - MQRO_EXPIRATION_WITH_FULL_DATA for expiration report

Call the value A.

On z/OS, the MQRO_*_WITH_DATA values should be used instead of the MQRO_*_WITH_FULL_DATA values.

2. Combine the *Report* field with A using the bitwise AND operation; call the result B.
3. Test B for equality with each of the values that is possible for that type of report.

For example, if A is MQRO_EXCEPTION_WITH_FULL_DATA, test B for equality with each of the following to determine what was specified by the sender of the message:

```
MQRO_NONE
MQRO_EXCEPTION
MQRO_EXCEPTION_WITH_DATA
MQRO_EXCEPTION_WITH_FULL_DATA
```

The tests can be performed in whatever order is most convenient for the application logic.

A similar method can be used to test for the MQRO_PASS_MSG_ID or MQRO_PASS_CORREL_ID options; select as the value A whichever of these two constants is appropriate, and then proceed as described above.

Using arithmetic

If the programming language *does not* support bit operations, the following steps should be performed using integer arithmetic:

1. Select one of the following values, according to the type of report to be checked:
 - MQRO_COA for COA report
 - MQRO_COD for COD report
 - MQRO_EXCEPTION for exception report
 - MQRO_EXPIRATION for expiration report

Call the value A.

2. Divide the *Report* field by A; call the result B.
3. Divide B by 8; call the result C.
4. Multiply C by 8 and subtract from B; call the result D.
5. Multiply D by A; call the result E.
6. Test E for equality with each of the values that is possible for that type of report.

For example, if A is MQRO_EXCEPTION, test E for equality with each of the following to determine what was specified by the sender of the message:

```
MQRO_NONE
MQRO_EXCEPTION
```

Report options

MQRO_EXCEPTION_WITH_DATA
MQRO_EXCEPTION_WITH_FULL_DATA

The tests can be performed in whatever order is most convenient for the application logic.

The following pseudocode illustrates this technique for exception report messages:

```
A = MQRO_EXCEPTION
B = Report/A
C = B/8
D = B - C*8
E = D*A
```

A similar method can be used to test for the MQRO_PASS_MSG_ID or MQRO_PASS_CORREL_ID options; select as the value A whichever of these two constants is appropriate, and then proceed as described above, but replacing the value 8 in the steps above by the value 2.

Structure of the message-flags field

The *MsgFlags* field is a 32-bit integer that is divided into three separate subfields. These subfields identify:

- Message flags that are rejected if the local queue manager does not recognize them
- Message flags that are always accepted, even if the local queue manager does not recognize them
- Message flags that are accepted only if certain other conditions are satisfied

Note: All subfields in *MsgFlags* are reserved for use by the queue manager.

Each subfield is identified by a bit mask which has 1-bits in the positions corresponding to the subfield, and 0-bits elsewhere. The bits are numbered such that bit 0 is the most significant bit, and bit 31 the least significant bit. The following masks are defined to identify the subfields:

MQMF_REJECT_UNSUP_MASK

Mask for unsupported message flags that are rejected.

This mask identifies the bit positions within the *MsgFlags* field where message flags which are not supported by the local queue manager will cause the MQPUT or MQPUT1 call to fail with completion code MQCC_FAILED and reason code MQRC_MSG_FLAGS_ERROR.

This subfield occupies bit positions 20 through 31.

The following message flags are included in this subfield:

```
MQMF_LAST_MSG_IN_GROUP
MQMF_LAST_SEGMENT
MQMF_MSG_IN_GROUP
MQMF_SEGMENT
MQMF_SEGMENTATION_ALLOWED
MQMF_SEGMENTATION_INHIBITED
```

MQMF_ACCEPT_UNSUP_MASK

Mask for unsupported message flags that are accepted.

Report options

This mask identifies the bit positions within the *MsgFlags* field where message flags which are not supported by the local queue manager will nevertheless be accepted on the MQPUT or MQPUT1 calls. The completion code is MQCC_OK.

This subfield occupies bit positions 0 through 11.

MQMF_ACCEPT_UNSUP_IF_XMIT_MASK

Mask for unsupported message flags that are accepted only in certain circumstances.

This mask identifies the bit positions within the *MsgFlags* field where message flags which are not supported by the local queue manager will nevertheless be accepted on the MQPUT or MQPUT1 calls *provided* that both of the following conditions are satisfied:

- The message is destined for a remote queue manager.
- The application is not putting the message directly on a local transmission queue (that is, the queue identified by the *ObjectQMgrName* and *ObjectName* fields in the object descriptor specified on the MQOPEN or MQPUT1 call is not a local transmission queue).

Completion code MQCC_OK is returned if these conditions are satisfied, and MQCC_FAILED with reason code MQRC_MSG_FLAGS_ERROR if not.

This subfield occupies bit positions 12 through 19.

If there are flags specified in the *MsgFlags* field that the queue manager does not recognize, the queue manager checks each subfield in turn by using the bitwise AND operation to combine the *MsgFlags* field with the mask for that subfield. If the result of that operation is not zero, the completion code and reason codes described above are returned.

Object attributes

Appendix F. Data conversion

This appendix describes the interface to the data-conversion exit, and the processing performed by the queue manager when data conversion is required.

The data-conversion exit is invoked as part of the processing of the MQGET call in order to convert the application message data to the representation required by the receiving application. Conversion of the application message data is optional — it requires the MQGMO_CONVERT option to be specified on the MQGET call.

The following are described:

- The processing performed by the queue manager in response to the MQGMO_CONVERT option; see “Conversion processing”.
- Processing conventions used by the queue manager when processing a built-in format; these conventions are recommended for user-written exits too. See “Processing conventions” on page 583.
- Special considerations for the conversion of report messages; see “Conversion of report messages” on page 587.
- The parameters passed to the data-conversion exit; see “MQ_DATA_CONV_EXIT – Data conversion exit” on page 602.
- A call that can be used from the exit in order to convert character data between different representations; see “MQXCNVC – Convert characters” on page 595.
- The data-structure parameter which is specific to the exit; see “MQDXP – Data-conversion exit parameter” on page 589.

Conversion processing

The queue manager performs the following actions if the MQGMO_CONVERT option is specified on the MQGET call, and there is a message to be returned to the application:

1. If one or more of the following is true, no conversion is necessary:
 - The message data is already in the character set and encoding required by the application issuing the MQGET call. The application must set the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter of the MQGET call to the values required, prior to issuing the call.
 - The length of the message data is zero.
 - The length of the *Buffer* parameter of the MQGET call is zero.

In these cases the message is returned without conversion to the application issuing the MQGET call; the *CodedCharSetId* and *Encoding* values in the *MsgDesc* parameter are set to the values in the control information in the message, and the call completes with one of the following combinations of completion code and reason code:

Completion code	Reason code
MQCC_OK	MQRC_NONE
MQCC_WARNING	MQRC_TRUNCATED_MSG_ACCEPTED
MQCC_WARNING	MQRC_TRUNCATED_MSG_FAILED

Conversion processing

The following steps are performed only if the character set or encoding of the message data differs from the corresponding value in the *MsgDesc* parameter, and there is data to be converted:

2. If the *Format* field in the control information in the message has the value MQFMT_NONE, the message is returned unconverted, with completion code MQCC_WARNING and reason code MQRC_FORMAT_ERROR.
In all other cases conversion processing continues.
3. The message is removed from the queue and placed in a temporary buffer which is the same size as the *Buffer* parameter. For browse operations, the message is copied into the temporary buffer, instead of being removed from the queue.
4. If the message has to be truncated to fit in the buffer, the following is done:
 - If the MQGMO_ACCEPT_TRUNCATED_MSG option was *not* specified, the message is returned unconverted, with completion code MQCC_WARNING and reason code MQRC_TRUNCATED_MSG_FAILED.
 - If the MQGMO_ACCEPT_TRUNCATED_MSG option *was* specified, the completion code is set to MQCC_WARNING, the reason code is set to MQRC_TRUNCATED_MSG_ACCEPTED, and conversion processing continues.
5. If the message can be accommodated in the buffer without truncation, or the MQGMO_ACCEPT_TRUNCATED_MSG option was specified, the following is done:
 - If the format is a built-in format, the buffer is passed to the queue-manager's data-conversion service.
 - If the format is not a built-in format, the buffer is passed to a user-written exit which has the same name as the format. If the exit cannot be found, the message is returned unconverted, with completion code MQCC_WARNING and reason code MQRC_FORMAT_ERROR.

If no error occurs, the output from the data-conversion service or from the user-written exit is the converted message, plus the completion code and reason code to be returned to the application issuing the MQGET call.

6. If the conversion is successful, the queue manager returns the converted message to the application. In this case, the completion code and reason code returned by the MQGET call will usually be one of the following combinations:

Completion code	Reason code
MQCC_OK	MQRC_NONE
MQCC_WARNING	MQRC_TRUNCATED_MSG_ACCEPTED

However, if the conversion is performed by a user-written exit, other reason codes can be returned, even when the conversion is successful.

If the conversion fails (for whatever reason), the queue manager returns the unconverted message to the application, with the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter set to the values in the control information in the message, and with completion code MQCC_WARNING. See below for possible reason codes.

Processing conventions

When converting a built-in format, the queue manager follows the processing conventions described below. It is recommended that user-written exits should also follow these conventions, although this is not enforced by the queue manager. The built-in formats converted by the queue manager are:

MQFMT_ADMIN
 MQFMT_CICS (z/OS only)
 MQFMT_COMMAND_1
 MQFMT_COMMAND_2
 MQFMT_DEAD_LETTER_HEADER
 MQFMT_DIST_HEADER
 MQFMT_EVENT version 1
 MQFMT_EVENT version 2 (z/OS only)
 MQFMT_IMS
 MQFMT_IMS_VAR_STRING
 MQFMT_MD_EXTENSION
 MQFMT_PCF
 MQFMT_REF_MSG_HEADER
 MQFMT_RF_HEADER
 MQFMT_RF_HEADER_2
 MQFMT_STRING
 MQFMT_TRIGGER
 MQFMT_WORK_INFO_HEADER (z/OS only)
 MQFMT_XMIT_Q_HEADER

1. If the message expands during conversion, and exceeds the size of the *Buffer* parameter, the following is done:
 - If the MQGMO_ACCEPT_TRUNCATED_MSG option was *not* specified, the message is returned unconverted, with completion code MQCC_WARNING and reason code MQRC_CONVERTED_MSG_TOO_BIG.
 - If the MQGMO_ACCEPT_TRUNCATED_MSG option *was* specified, the message is truncated, the completion code is set to MQCC_WARNING, the reason code is set to MQRC_TRUNCATED_MSG_ACCEPTED, and conversion processing continues.
2. If truncation occurs (either before or during conversion), it is possible for the number of valid bytes returned in the *Buffer* parameter to be *less than* the length of the buffer.

This can occur, for example, if a 4-byte integer or a DBCS character straddles the end of the buffer. The incomplete element of information is not converted, and so those bytes in the returned message do not contain valid information. This can also occur if a message that was truncated before conversion shrinks during conversion.

If the number of valid bytes returned is less than the length of the buffer, the unused bytes at the end of the buffer are set to nulls.
3. If an array or string straddles the end of the buffer, as much of the data as possible is converted; only the particular array element or DBCS character which is incomplete is not converted – preceding array elements or characters are converted.
4. If truncation occurs (either before or during conversion), the length returned for the *DataLength* parameter is the length of the *unconverted* message before truncation.
5. When strings are converted between single-byte character sets (SBCS), double-byte character sets (DBCS), or multi-byte character sets (MBCS), the strings can expand or contract.

Processing conventions

- In the PCF formats MQFMT_ADMIN, MQFMT_EVENT, and MQFMT_PCF, the strings in the MQCFST and MQCFSL structures expand or contract as necessary to accommodate the string after conversion.

For the string-list structure MQCFSL, the strings in the list may expand or contract by different amounts. If this happens, the queue manager pads the shorter strings with blanks to make them the same length as the longest string after conversion.

- In the format MQFMT_REF_MSG_HEADER, the strings addressed by the *SrcEnvOffset*, *SrcNameOffset*, *DestEnvOffset*, and *DestNameOffset* fields expand or contract as necessary to accommodate the strings after conversion.
 - In the format MQFMT_RF_HEADER, the *NameValueString* field expands or contracts as necessary to accommodate the name/value pairs after conversion.
 - In structures with fixed field sizes, the queue manager allows strings to expand or contract within their fixed fields, provided that no significant information is lost. In this regard, trailing blanks and characters following the first null character in the field are treated as insignificant.
 - If the string expands, but only insignificant characters need to be discarded to accommodate the converted string in the field, the conversion succeeds and the call completes with MQCC_OK and reason code MQRC_NONE (assuming no other errors).
 - If the string expands, but the converted string requires significant characters to be discarded in order to fit in the field, the message is returned unconverted and the call completes with MQCC_WARNING and reason code MQRC_CONVERTED_STRING_TOO_BIG.
- Note:** Reason code MQRC_CONVERTED_STRING_TOO_BIG results in this case whether or not the MQGMO_ACCEPT_TRUNCATED_MSG option was specified.
- If the string contracts, the queue manager pads the string with blanks to the length of the field.

6. For messages consisting of one or more MQ header structures followed by user data, it is possible for one or more of the header structures to be converted, while the remainder of the message is not. However, (with two exceptions) the *CodedCharSetId* and *Encoding* fields in each header structure always correctly indicate the character set and encoding of the data that follows the header structure.

The two exceptions are the MQCIH and MQIIH structures, where the values in the *CodedCharSetId* and *Encoding* fields in those structures are not significant. For those structures, the data following the structure is in the same character set and encoding as the MQCIH or MQIIH structure itself.

7. If the *CodedCharSetId* or *Encoding* fields in the control information of the message being retrieved, or in the *MsgDesc* parameter, specify values which are undefined or not supported, the queue manager may ignore the error if the undefined or unsupported value does not need to be used in converting the message.

For example, if the *Encoding* field in the message specifies an unsupported float encoding, but the message contains only integer data, or contains floating-point data which does not require conversion (because the source and target float encodings are identical), the error may or may not be diagnosed.

If the error is diagnosed, the message is returned unconverted, with completion code MQCC_WARNING and one of the

Processing conventions

MQRC_SOURCE_*_ERROR or MQRC_TARGET_*_ERROR reason codes (as appropriate); the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to the values in the control information in the message.

If the error is not diagnosed and the conversion completes successfully, the values returned in the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are those specified by the application issuing the MQGET call.

8. In all cases, if the message is returned to the application unconverted the completion code is set to MQCC_WARNING, and the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to the values appropriate to the unconverted data. This is done for MQFMT_NONE also.

The *Reason* parameter is set to a code that indicates why the conversion could not be carried out, unless the message also had to be truncated; reason codes related to truncation take precedence over reason codes related to conversion. (To determine if a truncated message was converted, check the values returned in the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter.)

When an error is diagnosed, either a specific reason code is returned, or the general reason code MQRC_NOT_CONVERTED. The reason code returned depends on the diagnostic capabilities of the underlying data-conversion service.

9. If completion code MQCC_WARNING is returned, and more than one reason code is relevant, the order of precedence is as follows:
 - a. The following reasons take precedence over all others; only one of the reasons in this group can arise:
 - MQRC_SIGNAL_REQUEST_ACCEPTED
 - MQRC_TRUNCATED_MSG_ACCEPTED
 - b. The order of precedence within the remaining reason codes is not defined.
10. On completion of the MQGET call:
 - The following reason code indicates that the message was converted successfully:
 - MQRC_NONE
 - The following reason code indicates that the message *may* have been converted successfully (check the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter to find out):
 - MQRC_TRUNCATED_MSG_ACCEPTED
 - All other reason codes indicate that the message was not converted.

The following processing is specific to the built-in formats; it is not applicable to user-defined formats:

11. With the exception of the following formats:
 - MQFMT_ADMIN
 - MQFMT_COMMAND_1
 - MQFMT_COMMAND_2
 - MQFMT_EVENT
 - MQFMT_IMS_VAR_STRING
 - MQFMT_PCF
 - MQFMT_STRING

none of the built-in formats can be converted from or to character sets that do not have SBCS characters for the characters that are valid in queue names. If an attempt is made to perform such a conversion, the message is returned unconverted, with completion code MQCC_WARNING and reason code MQRC_SOURCE_CCSD_ERROR or MQRC_TARGET_CCSD_ERROR, as appropriate.

Processing conventions

The Unicode character set UCS-2 is an example of a character set that does not have SBCS characters for the characters that are valid in queue names.

12. If the message data for a built-in format is truncated, fields within the message which contain lengths of strings, or counts of elements or structures, are *not* adjusted to reflect the length of the data actually returned to the application; the values returned for such fields within the message data are the values applicable to the message *prior to truncation*.

When processing messages such as a truncated MQFMT_ADMIN message, care must be taken to ensure that the application does not attempt to access data beyond the end of the data returned.

13. If the format name is MQFMT_DEAD_LETTER_HEADER, the message data begins with an MQDLH structure, and this may be followed by zero or more bytes of application message data. The format, character set, and encoding of the application message data are defined by the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQDLH structure at the start of the message. Since the MQDLH structure and application message data can have different character sets and encodings, it is possible for one, other, or both of the MQDLH structure and application message data to require conversion.

The queue manager converts the MQDLH structure first, as necessary. If conversion is successful, or the MQDLH structure does not require conversion, the queue manager checks the *CodedCharSetId* and *Encoding* fields in the MQDLH structure to see if conversion of the application message data is required. If conversion *is* required, the queue manager invokes the user-written exit with the name given by the *Format* field in the MQDLH structure, or performs the conversion itself (if *Format* is the name of a built-in format).

If the MQGET call returns a completion code of MQCC_WARNING, and the reason code is one of those indicating that conversion was not successful, one of the following applies:

- The MQDLH structure could not be converted. In this case the application message data will not have been converted either.
- The MQDLH structure was converted, but the application message data was not.

The application can examine the values returned in the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter, and those in the MQDLH structure, in order to determine which of the above applies.

14. If the format name is MQFMT_XMIT_Q_HEADER, the message data begins with an MQXQH structure, and this may be followed by zero or more bytes of additional data. This additional data is usually the application message data (which may be of zero length), but there can also be one or more further MQ header structures present, at the start of the additional data.

The MQXQH structure must be in the character set and encoding of the queue manager. The format, character set, and encoding of the data following the MQXQH structure are given by the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQMD structure contained *within* the MQXQH. For each subsequent MQ header structure present, the *Format*, *CodedCharSetId*, and *Encoding* fields in the structure describe the data that follows that structure; that data is either another MQ header structure, or the application message data.

If the MQGMO_CONVERT option is specified for an MQFMT_XMIT_Q_HEADER message, the application message data and certain of the MQ header structures are converted, *but the data in the MQXQH structure is not*. On return from the MQGET call, therefore:

- The values of the *Format*, *CodedCharSetId*, and *Encoding* fields in the *MsgDesc* parameter describe the data in the MQXQH structure, and *not* the application message data; the values will therefore *not* be the same as those specified by the application that issued the MQGET call.

The effect of this is that an application which repeatedly gets messages from a transmission queue with the MQGMO_CONVERT option specified must reset the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter to the values desired for the application message data, prior to each MQGET call.

- The values of the *Format*, *CodedCharSetId*, and *Encoding* fields in the last MQ header structure present describe the application message data. If there are no other MQ header structures present, the application message data is described by these fields in the MQMD structure within the MQXQH structure. If conversion is successful, the values will be the same as those specified in the *MsgDesc* parameter by the application that issued the MQGET call.

If the message is a distribution-list message, the MQXQH structure is followed by an MQDH structure (plus its arrays of MQOR and MQPMR records), which in turn may be followed by zero or more further MQ header structures and zero or more bytes of application message data. Like the MQXQH structure, the MQDH structure must be in the character set and encoding of the queue manager, and it is not converted on the MQGET call, even if the MQGMO_CONVERT option is specified.

The processing of the MQXQH and MQDH structures described above is primarily intended for use by message channel agents when they get messages from transmission queues.

Conversion of report messages

A report message can contain varying amounts of application message data, according to the report options specified by the sender of the original message. In particular, a report message can contain either:

1. No application message data
2. Some of the application message data from the original message
This occurs when the sender of the original message specifies MQRO_*_WITH_DATA and the message is longer than 100 bytes.
3. All of the application message data from the original message
This occurs when the sender of the original message specifies MQRO_*_WITH_FULL_DATA, or specifies MQRO_*_WITH_DATA and the message is 100 bytes or shorter.

When the queue manager or message channel agent generates a report message, it copies the format name from the original message into the *Format* field in the control information in the report message. The format name in the report message may therefore imply a length of data which is different from the length actually present in the report message (cases 1 and 2 above).

If the MQGMO_CONVERT option is specified when the report message is retrieved:

- For case 1 above, the data-conversion exit will not be invoked (because the report message will have no data).

Report message conversion

- For case 3 above, the format name correctly implies the length of the message data.
- But for case 2 above, the data-conversion exit will be invoked to convert a message which is *shorter* than the length implied by the format name.

In addition, the reason code passed to the exit will usually be MQRC_NONE (that is, the reason code will not indicate that the message has been truncated). This happens because the message data was truncated by the *sender* of the report message, and not by the receiver's queue manager in response to the MQGET call.

Because of these possibilities, the data-conversion exit should *not* use the format name to deduce the length of data passed to it; instead the exit should check the length of data provided, and be prepared to convert *less* data than the length implied by the format name. If the data can be converted successfully, completion code MQCC_OK and reason code MQRC_NONE should be returned by the exit. The length of the message data to be converted is passed to the exit as the *InBufferLength* parameter.

MQDXP – Data-conversion exit parameter

The following table summarizes the fields in the structure.

Table 100. Fields in MQDXP

Field	Description	Page
<i>StrucId</i>	Structure identifier	594
<i>Version</i>	Structure version number	594
<i>AppOptions</i>	Application options	590
<i>Encoding</i>	Numeric encoding required by application	591
<i>CodedCharSetId</i>	Character set required by application	590
<i>DataLength</i>	Length in bytes of message data	590
<i>CompCode</i>	Completion code	590
<i>Reason</i>	Reason code qualifying <i>CompCode</i>	592
<i>ExitResponse</i>	Response from exit	591
<i>Hconn</i>	Connection handle	592

Overview

Availability: Not VSE/ESA, Windows 3.1, Windows 95, Windows 98.

Purpose: The MQDXP structure is a parameter that the queue manager passes to the data-conversion exit when the exit is invoked to convert the message data as part of the processing of the MQGET call. See the description of the MQ_DATA_CONV_EXIT call for details of the data conversion exit.

Character set and encoding: Character data in MQDXP is in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute. Numeric data in MQDXP is in the native machine encoding; this is given by MQENC_NATIVE.

Usage: Only the *DataLength*, *CompCode*, *Reason* and *ExitResponse* fields in MQDXP may be changed by the exit; changes to other fields are ignored. However, the *DataLength* field *cannot* be changed if the message being converted is a segment that contains only part of a logical message.

When control returns to the queue manager from the exit, the queue manager checks the values returned in MQDXP. If the values returned are not valid, the queue manager continues processing as though the exit had returned MQXDR_CONVERSION_FAILED in *ExitResponse*; however, the queue manager ignores the values of the *CompCode* and *Reason* fields returned by the exit in this case, and uses instead the values those fields had on *input* to the exit. The following values in MQDXP cause this processing to occur:

- *ExitResponse* field not MQXDR_OK and not MQXDR_CONVERSION_FAILED
- *CompCode* field not MQCC_OK and not MQCC_WARNING
- *DataLength* field less than zero, or *DataLength* field changed when the message being converted is a segment that contains only part of a logical message.

MQDXP – Data-conversion exit parameter

Fields

The MQDXP structure contains the following fields; the fields are described in **alphabetic order**:

AppOptions (MQLONG)

Application options.

This is a copy of the *Options* field of the MQGMO structure specified by the application issuing the MQGET call. The exit may need to examine these to ascertain whether the MQGMO_ACCEPT_TRUNCATED_MSG option was specified.

This is an input field to the exit.

CodedCharSetId (MQLONG)

Character set required by application.

This is the coded character-set identifier of the character set required by the application issuing the MQGET call; see the *CodedCharSetId* field in the MQMD structure for more details. If the application specifies the special value MQCCSI_Q_MGR on the MQGET call, the queue manager changes this to the actual character-set identifier of the character set used by the queue manager, before invoking the exit.

If the conversion is successful, the exit should copy this to the *CodedCharSetId* field in the message descriptor.

This is an input field to the exit.

CompCode (MQLONG)

Completion code.

When the exit is invoked, this contains the completion code that will be returned to the application that issued the MQGET call, if the exit chooses to do nothing. It is always MQCC_WARNING, because either the message was truncated, or the message requires conversion and this has not yet been done.

On output from the exit, this field contains the completion code to be returned to the application in the *CompCode* parameter of the MQGET call; only MQCC_OK and MQCC_WARNING are valid. See the description of the *Reason* field for recommendations on how the exit should set this field on output.

This is an input/output field to the exit.

DataLength (MQLONG)

Length in bytes of message data.

When the exit is invoked, this field contains the original length of the application message data. If the message was truncated in order to fit into the buffer provided by the application, the size of the message provided to the exit will be *smaller* than the value of *DataLength*. The size of the message actually provided to the exit is always given by the *InBufferLength* parameter of the exit, irrespective of any truncation that may have occurred.

Truncation is indicated by the *Reason* field having the value MQRC_TRUNCATED_MSG_ACCEPTED on input to the exit.

Most conversions will not need to change this length, but an exit can do so if necessary; the value set by the exit is returned to the application in the *DataLength* parameter of the MQGET call. However, this length *cannot* be changed if the message being converted is a segment that contains only part of a logical message. This is because changing the length would cause the offsets of later segments in the logical message to be incorrect.

Note that, if the exit wants to change the length of the data, be aware that the queue manager has already decided whether the message data will fit into the application's buffer, based on the length of the *unconverted* data. This decision determines whether the message is removed from the queue (or the browse cursor moved, for a browse request), and is not affected by any change to the data length caused by the conversion. For this reason it is recommended that conversion exits do not cause a change in the length of the application message data.

If character conversion does imply a change of length, a string can be converted into another string with the same length in bytes, truncating trailing blanks or padding with blanks as necessary.

The exit is not invoked if the message contains no application message data; hence *DataLength* is always greater than zero.

This is an input/output field to the exit.

Encoding (MQLONG)

Numeric encoding required by application.

This is the numeric encoding required by the application issuing the MQGET call; see the *Encoding* field in the MQMD structure for more details.

If the conversion is successful, the exit should copy this to the *Encoding* field in the message descriptor.

This is an input field to the exit.

ExitOptions (MQLONG)

Reserved.

This is a reserved field; its value is 0.

ExitResponse (MQLONG)

Response from exit.

This is set by the exit to indicate the success or otherwise of the conversion. It must be one of the following:

MQXDR_OK

Conversion was successful.

If the exit specifies this value, the queue manager returns the following to the application that issued the MQGET call:

- The value of the *CompCode* field on output from the exit
- The value of the *Reason* field on output from the exit
- The value of the *DataLength* field on output from the exit
- The contents of the exit's output buffer *OutBuffer*. The number of bytes returned is the lesser of the exit's *OutBufferLength* parameter, and the value of the *DataLength* field on output from the exit

MQDXP – ExitResponse field

If the *Encoding* and *CodedCharSetId* fields in the exit's message descriptor parameter are *both* unchanged, the queue manager returns:

- The value of the *Encoding* and *CodedCharSetId* fields in the MQDXP structure on *input* to the exit

If one or both of the *Encoding* and *CodedCharSetId* fields in the exit's message descriptor parameter has been changed, the queue manager returns:

- The value of the *Encoding* and *CodedCharSetId* fields in the exit's message descriptor parameter on output from the exit

MQXDR_CONVERSION_FAILED

Conversion was unsuccessful.

If the exit specifies this value, the queue manager returns the following to the application that issued the MQGET call:

- The value of the *CompCode* field on output from the exit
- The value of the *Reason* field on output from the exit
- The value of the *DataLength* field on *input* to the exit
- The contents of the exit's input buffer *InBuffer*. The number of bytes returned is given by the *InBufferLength* parameter

If the exit has altered *InBuffer*, the results are undefined.

ExitResponse is an output field from the exit.

Hconn (MQHCONN)

Connection handle.

This is a connection handle which can be used on the MQXCNVC call. This handle is not necessarily the same as the handle specified by the application which issued the MQGET call.

Reason (MQLONG)

Reason code qualifying *CompCode*.

When the exit is invoked, this contains the reason code that will be returned to the application that issued the MQGET call, if the exit chooses to do nothing. Among possible values are MQRC_TRUNCATED_MSG_ACCEPTED, indicating that the message was truncated in order fit into the buffer provided by the application, and MQRC_NOT_CONVERTED, indicating that the message requires conversion but that this has not yet been done.

On output from the exit, this field contains the reason to be returned to the application in the *Reason* parameter of the MQGET call; the following is recommended:

- If *Reason* had the value MQRC_TRUNCATED_MSG_ACCEPTED on input to the exit, the *Reason* and *CompCode* fields should not be altered, irrespective of whether the conversion succeeds or fails.

(If the *CompCode* field is not MQCC_OK, the application which retrieves the message can identify a conversion failure by comparing the returned *Encoding* and *CodedCharSetId* values in the message descriptor with the values requested; in contrast, the application cannot distinguish a truncated message from a message that just fitted the buffer. For this reason,

MQRC_TRUNCATED_MSG_ACCEPTED should be returned in preference to any of the reasons that indicate conversion failure.)

- If *Reason* had any other value on input to the exit:
 - If the conversion succeeds, *CompCode* should be set to MQCC_OK and *Reason* set to MQRC_NONE.
 - If the conversion fails, or the message expands and has to be truncated to fit in the buffer, *CompCode* should be set to MQCC_WARNING (or left unchanged), and *Reason* set to one of the values listed below, to indicate the nature of the failure.

Note that, if the message after conversion is too big for the buffer, it should be truncated only if the application that issued the MQGET call specified the MQGMO_ACCEPT_TRUNCATED_MSG option:

- If it did specify that option, reason MQRC_TRUNCATED_MSG_ACCEPTED should be returned.
- If it did not specify that option, the message should be returned unconverted, with reason code MQRC_CONVERTED_MSG_TOO_BIG.

The reason codes listed below are recommended for use by the exit to indicate the reason that conversion failed, but the exit can return other values from the set of MQRC_* codes if deemed appropriate. In addition, the range of values MQRC_APPL_FIRST through MQRC_APPL_LAST are allocated for use by the exit to indicate conditions that the exit wishes to communicate to the application issuing the MQGET call.

Note: If the message cannot be converted successfully, the exit *must* return MQXDR_CONVERSION_FAILED in the *ExitResponse* field, in order to cause the queue manager to return the unconverted message. This is true regardless of the reason code returned in the *Reason* field.

MQRC_APPL_FIRST

(900, X'384') Lowest value for application-defined reason code.

MQRC_APPL_LAST

(999, X'3E7') Highest value for application-defined reason code.

MQRC_CONVERTED_MSG_TOO_BIG

(2120, X'848') Converted data too big for buffer.

MQRC_NOT_CONVERTED

(2119, X'847') Message data not converted.

MQRC_SOURCE_CCSD_ERROR

(2111, X'83F') Source coded character set identifier not valid.

MQRC_SOURCE_DECIMAL_ENC_ERROR

(2113, X'841') Packed-decimal encoding in message not recognized.

MQRC_SOURCE_FLOAT_ENC_ERROR

(2114, X'842') Floating-point encoding in message not recognized.

MQRC_SOURCE_INTEGER_ENC_ERROR

(2112, X'840') Source integer encoding not recognized.

MQRC_TARGET_CCSD_ERROR

(2115, X'843') Target coded character set identifier not valid.

MQRC_TARGET_DECIMAL_ENC_ERROR

(2117, X'845') Packed-decimal encoding specified by receiver not recognized.

MQRC_TARGET_FLOAT_ENC_ERROR

(2118, X'846') Floating-point encoding specified by receiver not recognized.

MQRC_TARGET_INTEGER_ENC_ERROR

(2116, X'844') Target integer encoding not recognized.

MQDXP – Reason field

MQRC_TRUNCATED_MSG_ACCEPTED
(2079, X'81F') Truncated message returned (processing completed).

This is an input/output field to the exit.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQDXP_STRUC_ID

Identifier for data conversion exit parameter structure.

For the C programming language, the constant `MQDXP_STRUC_ID_ARRAY` is also defined; this has the same value as `MQDXP_STRUC_ID`, but is an array of characters instead of a string.

This is an input field to the exit.

Version (MQLONG)

Structure version number.

The value must be:

MQDXP_VERSION_1

Version number for data-conversion exit parameter structure.

The following constant specifies the version number of the current version:

MQDXP_CURRENT_VERSION

Current version of data-conversion exit parameter structure.

Note: When a new version of this structure is introduced, the layout of the existing part is not changed. The exit should therefore check that the *Version* field is equal to or greater than the lowest version which contains the fields that the exit needs to use.

This is an input field to the exit.

C declaration

```
typedef struct tagMQDXP MQDXP;
struct tagMQDXP {
    MQCHAR4  StrucId;           /* Structure identifier */
    MQLONG   Version;          /* Structure version number */
    MQLONG   ExitOptions;      /* Reserved */
    MQLONG   AppOptions;       /* Application options */
    MQLONG   Encoding;         /* Numeric encoding required by
                               application */
    MQLONG   CodedCharSetId;   /* Character set required by application */
    MQLONG   DataLength;       /* Length in bytes of message data */
    MQLONG   CompCode;         /* Completion code */
    MQLONG   Reason;           /* Reason code qualifying CompCode */
    MQLONG   ExitResponse;     /* Response from exit */
    MQHCONN  Hconn;           /* Connection handle */
};
```

COBOL declaration (OS/400 only)

```
** MQDXP structure
10 MQDXP.
** Structure identifier
```

MQDXP – Language declarations

```
15 MQDXP-STRUCID      PIC X(4).
**   Structure version number
15 MQDXP-VERSION      PIC S9(9) BINARY.
**   Reserved
15 MQDXP-EXITOPTIONS  PIC S9(9) BINARY.
**   Application options
15 MQDXP-APPOPTIONS  PIC S9(9) BINARY.
**   Numeric encoding required by application
15 MQDXP-ENCODING     PIC S9(9) BINARY.
**   Character set required by application
15 MQDXP-CODEDCHARSETID PIC S9(9) BINARY.
**   Length in bytes of message data
15 MQDXP-DATALength  PIC S9(9) BINARY.
**   Completion code
15 MQDXP-COMPCODE     PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
15 MQDXP-REASON       PIC S9(9) BINARY.
**   Response from exit
15 MQDXP-EXITRESPONSE PIC S9(9) BINARY.
**   Connection handle
15 MQDXP-HCONN        PIC S9(9) BINARY.
```

System/390 assembler declaration

```
MQDXP          DSECT
MQDXP_STRUCID  DS   CL4  Structure identifier
MQDXP_VERSION  DS   F    Structure version number
MQDXP_EXITOPTIONS DS  F    Reserved
MQDXP_APPOPTIONS DS  F    Application options
MQDXP_ENCODING DS  F    Numeric encoding required by application
MQDXP_CODEDCHARSETID DS  F    Character set required by application
MQDXP_DATALength DS  F    Length in bytes of message data
MQDXP_COMPCODE DS  F    Completion code
MQDXP_REASON   DS  F    Reason code qualifying COMPCODE
MQDXP_EXITRESPONSE DS  F    Response from exit
MQDXP_HCONN    DS  F    Connection handle
*
MQDXP_LENGTH   EQU  *-MQDXP
ORG  MQDXP
MQDXP_AREA     DS   CL(MQDXP_LENGTH)
```

MQXCNVC – Convert characters

The MQXCNVC call converts characters from one character set to another.

- On z/OS, the call can be used from an application as well as from a data-conversion exit.
- In other environments, the call can be used only from a data-conversion exit.

This call is part of the MQSeries Data Conversion Interface (DCI), which is one of the MQSeries framework interfaces. Note: this call can be used only from a data-conversion exit.

Syntax

```
MQXCNVC (Hconn, Options, SourceCCSID, SourceLength, SourceBuffer,  
         TargetCCSID, TargetLength, TargetBuffer, DataLength, CompCode, Reason)
```

Parameters

The MQXCNVC call has the following parameters.

MQXCNVC – Hconn parameter

Hconn (MQHCONN) – input

Connection handle.

This handle represents the connection to the queue manager.

In a data-conversion exit, *Hconn* should normally be the handle passed to the data-conversion exit in the *Hconn* field of the MQDXP structure; this handle is not necessarily the same as the handle specified by the application which issued the MQGET call.

On OS/400, the following special value can be specified for *Hconn*:

MQHC_DEF_HCONN

Default connection handle.

Options (MQLONG) – input

Options that control the action of MQXCNVC.

Zero or more of the options described below can be specified. If more than one is required, the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations)

Default-conversion option: The following option controls the use of default character conversion:

MQDCC_DEFAULT_CONVERSION

Default conversion.

This option specifies that default character conversion can be used if one or both of the character sets specified on the call is not supported. This allows the queue manager to use an installation-specified default character set that approximates the specified character set, when converting the string.

Note: The result of using an approximate character set to convert the string is that some characters may be converted incorrectly. This can be avoided by using in the string only characters which are common to both the specified character set and the default character set.

The default character sets are defined by a configuration option when the queue manager is installed or restarted.

If MQDCC_DEFAULT_CONVERSION is not specified, the queue manager uses only the specified character sets to convert the string, and the call fails if one or both of the character sets is not supported.

This option is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Solaris, Linux, Windows.

Padding option: The following option allows the queue manager to pad the converted string with blanks or discard insignificant trailing characters, in order to make the converted string fit the target buffer:

MQDCC_FILL_TARGET_BUFFER

Fill target buffer.

MQXCNVC – Options parameter

This option requests that conversion take place in such a way that the target buffer is filled completely:

- If the string contracts when it is converted, trailing blanks are added in order to fill the target buffer.
- If the string expands when it is converted, trailing characters that are not significant are discarded to make the converted string fit the target buffer. If this can be done successfully, the call completes with MQCC_OK and reason code MQRC_NONE.

If there are too few insignificant trailing characters, as much of the string as will fit is placed in the target buffer, and the call completes with MQCC_WARNING and reason code MQRC_CONVERTED_MSG_TOO_BIG.

Insignificant characters are:

- Trailing blanks
- Characters following the first null character in the string (but excluding the first null character itself)
- If the string, *TargetCCSID*, and *TargetLength* are such that the target buffer cannot be set completely with valid characters, the call fails with MQCC_FAILED and reason code MQRC_TARGET_LENGTH_ERROR. This can occur when *TargetCCSID* is a pure DBCS character set (such as UCS-2), but *TargetLength* specifies a length that is an odd number of bytes.
- *TargetLength* can be less than or greater than *SourceLength*. On return from MQXCNVC, *DataLength* has the same value as *TargetLength*.

If this option is not specified:

- The string is allowed to contract or expand within the target buffer as required. Insignificant trailing characters are neither added nor discarded.

If the converted string fits in the target buffer, the call completes with MQCC_OK and reason code MQRC_NONE.

If the converted string is too big for the target buffer, as much of the string as will fit is placed in the target buffer, and the call completes with MQCC_WARNING and reason code MQRC_CONVERTED_MSG_TOO_BIG. Note that fewer than *TargetLength* bytes can be returned in this case.

- *TargetLength* can be less than or greater than *SourceLength*. On return from MQXCNVC, *DataLength* is less than or equal to *TargetLength*.

This option is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Solaris, Linux, Windows.

Encoding options: The options described below can be used to specify the integer encodings of the source and target strings. The relevant encoding is used *only* when the corresponding character set identifier indicates that the representation of the character set in main storage is dependent on the encoding used for binary integers. This affects only certain multibyte character sets (for example, UCS-2 character sets).

The encoding is ignored if the character set is a single-byte character set (SBCS), or a multibyte character set whose representation in main storage is not dependent on the integer encoding.

MQXCNVC – Options parameter

Only one of the MQDCC_SOURCE_* values should be specified, combined with one of the MQDCC_TARGET_* values:

MQDCC_SOURCE_ENC_NATIVE

Source encoding is the default for the environment and programming language.

MQDCC_SOURCE_ENC_NORMAL

Source encoding is normal.

MQDCC_SOURCE_ENC_REVERSED

Source encoding is reversed.

MQDCC_SOURCE_ENC_UNDEFINED

Source encoding is undefined.

MQDCC_TARGET_ENC_NATIVE

Target encoding is the default for the environment and programming language.

MQDCC_TARGET_ENC_NORMAL

Target encoding is normal.

MQDCC_TARGET_ENC_REVERSED

Target encoding is reversed.

MQDCC_TARGET_ENC_UNDEFINED

Target encoding is undefined.

The encoding values defined above can be added directly to the *Options* field. However, if the source or target encoding is obtained from the *Encoding* field in the MQMD or other structure, the following processing must be done:

1. The integer encoding must be extracted from the *Encoding* field by eliminating the float and packed-decimal encodings; see “Analyzing encodings” on page 573 for details of how to do this.
2. The integer encoding resulting from step 1 must be multiplied by the appropriate factor before being added to the *Options* field. These factors are:
 - MQDCC_SOURCE_ENC_FACTOR for the source encoding
 - MQDCC_TARGET_ENC_FACTOR for the target encoding

The following illustrates how this might be coded in the C programming language:

```
Options = (MsgDesc.Encoding & MQENC_INTEGER_MASK)
          * MQDCC_SOURCE_ENC_FACTOR
          + (DataConvExitParms.Encoding & MQENC_INTEGER_MASK)
          * MQDCC_TARGET_ENC_FACTOR;
```

If not specified, the encoding options default to undefined (MQDCC*_ENC_UNDEFINED). In most cases, this does not affect the successful completion of the MQXCNVC call. However, if the corresponding character set is a multibyte character set whose representation is dependent on the encoding (for example, a UCS-2 character set), the call fails with reason code MQRC_SOURCE_INTEGER_ENC_ERROR or MQRC_TARGET_INTEGER_ENC_ERROR as appropriate.

The encoding options are supported in the following environments: AIX, HP-UX, z/OS, OS/2, OS/400, Solaris, Linux, Windows.

Default option: If none of the options described above is specified, the following option can be used:

MQDCC_NONE

No options specified.

MQDCC_NONE is defined to aid program documentation. It is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

SourceCCSID (MQLONG) – input

Coded character set identifier of string before conversion.

This is the coded character set identifier of the input string in *SourceBuffer*.

SourceLength (MQLONG) – input

Length of string before conversion.

This is the length in bytes of the input string in *SourceBuffer*; it must be zero or greater.

SourceBuffer (MQCHAR×SourceLength) – input

String to be converted.

This is the buffer containing the string to be converted from one character set to another.

TargetCCSID (MQLONG) – input

Coded character set identifier of string after conversion.

This is the coded character set identifier of the character set to which *SourceBuffer* is to be converted.

TargetLength (MQLONG) – input

Length of output buffer.

This is the length in bytes of the output buffer *TargetBuffer*; it must be zero or greater. It can be less than or greater than *SourceLength*.

TargetBuffer (MQCHAR×TargetLength) – output

String after conversion.

This is the string after it has been converted to the character set defined by *TargetCCSID*. The converted string can be shorter or longer than the unconverted string. The *DataLength* parameter indicates the number of valid bytes returned.

DataLength (MQLONG) – output

Length of output string.

This is the length of the string returned in the output buffer *TargetBuffer*. The converted string can be shorter or longer than the unconverted string.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_WARNING

Warning (partial completion).

MQXCNCV – CompCode parameter

MQCC_FAILED
Call failed.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_CONVERTED_MSG_TOO_BIG
(2120, X'848') Converted data too big for buffer.

If *CompCode* is MQCC_FAILED:

MQRC_DATA_LENGTH_ERROR
(2010, X'7DA') Data length parameter not valid.

MQRC_DBCS_ERROR
(2150, X'866') DBCS string not valid.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SOURCE_BUFFER_ERROR
(2145, X'861') Source buffer parameter not valid.

MQRC_SOURCE_CCSID_ERROR
(2111, X'83F') Source coded character set identifier not valid.

MQRC_SOURCE_INTEGER_ENC_ERROR
(2112, X'840') Source integer encoding not recognized.

MQRC_SOURCE_LENGTH_ERROR
(2143, X'85F') Source length parameter not valid.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_TARGET_BUFFER_ERROR
(2146, X'862') Target buffer parameter not valid.

MQRC_TARGET_CCSID_ERROR
(2115, X'843') Target coded character set identifier not valid.

MQRC_TARGET_INTEGER_ENC_ERROR
(2116, X'844') Target integer encoding not recognized.

MQRC_TARGET_LENGTH_ERROR
(2144, X'860') Target length parameter not valid.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For more information on these reason codes, see Appendix A, “Return codes”, on page 527.

C invocation

```
MQXCNCV (Hconn, Options, SourceCCSID, SourceLength, SourceBuffer,  
TargetCCSID, TargetLength, TargetBuffer, &DataLength,  
&CompCode, &Reason);
```

Declare the parameters as follows:

MQXCNCV – Language invocations

```
MQHCONN Hconn;          /* Connection handle */
MQLONG  Options;        /* Options that control the action of
                        MQXCNCV */
MQLONG  SourceCCSID;    /* Coded character set identifier of string
                        before conversion */
MQLONG  SourceLength;   /* Length of string before conversion */
MQCHAR  SourceBuffer[n]; /* String to be converted */
MQLONG  TargetCCSID;    /* Coded character set identifier of string
                        after conversion */
MQLONG  TargetLength;   /* Length of output buffer */
MQCHAR  TargetBuffer[n]; /* String after conversion */
MQLONG  DataLength;     /* Length of output string */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

COBOL invocation (OS/400 only)

```
CALL 'MQXCNCV' USING HCONN, OPTIONS, SOURCECCSID, SOURCELENGTH,
                    SOURCEBUFFER, TARGETCCSID, TARGETLENGTH,
                    TARGETBUFFER, DATALENGTH, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN          PIC S9(9) BINARY.
** Options that control the action of MQXCNCV
01 OPTIONS        PIC S9(9) BINARY.
** Coded character set identifier of string before conversion
01 SOURCECCSID    PIC S9(9) BINARY.
** Length of string before conversion
01 SOURCELENGTH   PIC S9(9) BINARY.
** String to be converted
01 SOURCEBUFFER    PIC X(n).
** Coded character set identifier of string after conversion
01 TARGETCCSID    PIC S9(9) BINARY.
** Length of output buffer
01 TARGETLENGTH   PIC S9(9) BINARY.
** String after conversion
01 TARGETBUFFER    PIC X(n).
** Length of output string
01 DATALENGTH    PIC S9(9) BINARY.
** Completion code
01 COMPCODE        PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON          PIC S9(9) BINARY.
```

System/390 assembler invocation

```
CALL MQXCNCV, (HCONN, OPTIONS, SOURCECCSID, SOURCELENGTH,      X
               SOURCEBUFFER, TARGETCCSID, TARGETLENGTH, TARGETBUFFER, X
               DATALENGTH, COMPCODE, REASON)
```

Declare the parameters as follows:

```
HCONN          DS F      Connection handle
OPTIONS        DS F      Options that control the action of MQXCNCV
SOURCECCSID    DS F      Coded character set identifier of string before
* conversion
SOURCELENGTH   DS F      Length of string before conversion
SOURCEBUFFER    DS CL(n) String to be converted
TARGETCCSID    DS F      Coded character set identifier of string after
* conversion
TARGETLENGTH   DS F      Length of output buffer
TARGETBUFFER    DS CL(n) String after conversion
DATALENGTH     DS F      Length of output string
COMPCODE       DS F      Completion code
REASON         DS F      Reason code qualifying COMPCODE
```

MQ_DATA_CONV_EXIT – Data conversion exit

This call definition describes the parameters that are passed to the data-conversion exit. No entry point called MQ_DATA_CONV_EXIT is actually provided by the queue manager (see usage note 11 on page 605).

This definition is part of the MQSeries Data Conversion Interface (DCI), which is one of the MQSeries framework interfaces.

Syntax

```
MQ_DATA_CONV_EXIT (DataConvExitParms, MsgDesc, InBufferLength,
                  InBuffer, OutBufferLength, OutBuffer)
```

Parameters

The MQ_DATA_CONV_EXIT call has the following parameters.

DataConvExitParms (MQDXP) – input/output

Data-conversion exit parameter block.

This structure contains information relating to the invocation of the exit. The exit sets information in this structure to indicate the outcome of the conversion. See “MQDXP – Data-conversion exit parameter” on page 589 for details of the fields in this structure.

MsgDesc (MQMD) – input/output

Message descriptor.

On input to the exit, this is the message descriptor associated with the message data passed to the exit in the *InBuffer* parameter.

Note: The *MsgDesc* parameter passed to the exit is always the most-recent version of MQMD supported by the queue manager which invokes the exit. If the exit is intended to be portable between different environments, the exit should check the *Version* field in *MsgDesc* to verify that the fields that the exit needs to access are present in the structure.

In the following environments, the exit is passed a version-2 MQMD: AIX, HP-UX, OS/2, OS/400, Solaris, Linux, Windows. In all other environments that support the data conversion exit, the exit is passed a version-1 MQMD.

On output, the exit should change the *Encoding* and *CodedCharSetId* fields to the values requested by the application, if conversion was successful; these changes will be reflected back to the application. Any other changes that the exit makes to the structure are ignored; they are not reflected back to the application.

If the exit returns MQXDR_OK in the *ExitResponse* field of the MQDXP structure, but does not change the *Encoding* or *CodedCharSetId* fields in the message descriptor, the queue manager returns for those fields the values that the corresponding fields in the MQDXP structure had on input to the exit.

InBufferLength (MQLONG) – input

Length in bytes of *InBuffer*.

MQ_DATA_CONV_EXIT – InBufferLength parameter

This is the length of the input buffer *InBuffer*, and specifies the number of bytes to be processed by the exit. *InBufferLength* is the lesser of the length of the message data prior to conversion, and the length of the buffer provided by the application on the MQGET call.

The value is always greater than zero.

InBuffer (MQBYTE×InBufferLength) – input

Buffer containing the unconverted message.

This contains the message data prior to conversion. If the exit is unable to convert the data, the queue manager returns the contents of this buffer to the application after the exit has completed.

Note: The exit should not alter *InBuffer*; if this parameter is altered, the results are undefined.

In the C programming language, this parameter is defined as a pointer-to-void.

OutBufferLength (MQLONG) – input

Length in bytes of *OutBuffer*.

This is the length of the output buffer *OutBuffer*, and is the same as the length of the buffer provided by the application on the MQGET call.

The value is always greater than zero.

OutBuffer (MQBYTE×OutBufferLength) – output

Buffer containing the converted message.

On output from the exit, if the conversion was successful (as indicated by the value MQXDR_OK in the *ExitResponse* field of the *DataConvExitParms* parameter), *OutBuffer* contains the message data to be delivered to the application, in the requested representation. If the conversion was unsuccessful, any changes that the exit has made to this buffer are ignored.

In the C programming language, this parameter is defined as a pointer-to-void.

Usage notes

1. A data-conversion exit is a user-written exit which receives control during the processing of an MQGET call. The function performed by the data-conversion exit is defined by the provider of the exit; however, the exit must conform to the rules described here, and in the associated parameter structure MQDXP. The programming languages that can be used for a data-conversion exit are determined by the environment.
2. The exit is invoked only if *all* of the following are true:
 - The MQGMO_CONVERT option is specified on the MQGET call
 - The *Format* field in the message descriptor is not MQFMT_NONE
 - The message is not already in the required representation; that is, one or both of the message's *CodedCharSetId* and *Encoding* is different from the value specified by the application in the message descriptor supplied on the MQGET call
 - The queue manager has not already done the conversion successfully
 - The length of the application's buffer is greater than zero

MQ_DATA_CONV_EXIT – Usage notes

- The length of the message data is greater than zero
 - The reason code so far during the MQGET operation is MQRC_NONE or MQRC_TRUNCATED_MSG_ACCEPTED
3. When an exit is being written, consideration should be given to coding the exit in a way that will allow it to convert messages that have been truncated. Truncated messages can arise in the following ways:
- The receiving application provides a buffer that is smaller than the message, but specifies the MQGMO_ACCEPT_TRUNCATED_MSG option on the MQGET call.
 - The sender of the message truncated it before sending it. This can happen with report messages, for example (see “Conversion of report messages” on page 587 for more details).

In this case, the *Reason* field in the *DataConvExitParms* parameter on input to the exit will have the value MQRC_TRUNCATED_MSG_ACCEPTED.

In this case, the *Reason* field in the *DataConvExitParms* parameter on input to the exit will have the value MQRC_NONE (if the receiving application provided a buffer that was big enough for the message).

Thus the value of the *Reason* field on input to the exit cannot always be used to decide whether the message has been truncated.

The distinguishing characteristic of a truncated message is that the length provided to the exit in the *InBufferLength* parameter will be *less than* the length implied by the format name contained in the *Format* field in the message descriptor. The exit should therefore check the value of *InBufferLength* before attempting to convert any of the data; the exit *should not* assume that the full amount of data implied by the format name has been provided.

If the exit has *not* been written to convert truncated messages, and *InBufferLength* is less than the value expected, the exit should return MQXDR_CONVERSION_FAILED in the *ExitResponse* field of the *DataConvExitParms* parameter, with the *CompCode* and *Reason* fields set to MQCC_WARNING and MQRC_FORMAT_ERROR respectively.

If the exit *has* been written to convert truncated messages, the exit should convert as much of the data as possible (see next usage note), taking care not to attempt to examine or convert data beyond the end of *InBuffer*. If the conversion completes successfully, the exit should leave the *Reason* field in the *DataConvExitParms* parameter unchanged. This has the effect of returning MQRC_TRUNCATED_MSG_ACCEPTED if the message was truncated by the receiver’s queue manager, and MQRC_NONE if the message was truncated by the sender of the message.

It is also possible for a message to expand *during* conversion, to the point where it is bigger than *OutBuffer*. In this case the exit must decide whether to truncate the message; the *AppOptions* field in the *DataConvExitParms* parameter will indicate whether the receiving application specified the MQGMO_ACCEPT_TRUNCATED_MSG option.

4. Generally it is recommended that all of the data in the message provided to the exit in *InBuffer* is converted, or that none of it is. An exception to this, however, occurs if the message is truncated, either before conversion or during conversion; in this case there may be an incomplete item at the end of the buffer (for example: one byte of a double-byte character, or 3 bytes of a 4-byte integer). In this situation it is recommended that the incomplete item

MQ_DATA_CONV_EXIT – Usage notes

should be omitted, and unused bytes in *OutBuffer* set to nulls. However, complete elements or characters within an array or string *should* be converted.

5. When an exit is needed for the first time, the queue manager attempts to load an object that has the same name as the format (apart from extensions). The object loaded must contain the exit that processes messages with that format name. It is recommended that the exit name, and the name of the object that contain the exit, should be identical, although not all environments require this.
6. A new copy of the exit is loaded when an application attempts to retrieve the first message that uses that *Format* since the application connected to the queue manager. For CICS or IMS applications, this means when the CICS or IMS subsystem connected to the queue manager. A new copy may also be loaded at other times, if the queue manager has discarded a previously-loaded copy. For this reason, an exit should not attempt to use static storage to communicate information from one invocation of the exit to the next – the exit may be unloaded between the two invocations.
7. If there is a user-supplied exit with the same name as one of the built-in formats supported by the queue manager, the user-supplied exit does not replace the built-in conversion routine. The only circumstances in which such an exit is invoked are:
 - If the built-in conversion routine cannot handle conversions to or from either the *CodedCharSetId* or *Encoding* involved, or
 - If the built-in conversion routine has failed to convert the data (for example, because there is a field or character which cannot be converted).
8. The scope of the exit is environment-dependent. *Format* names should be chosen so as to minimize the risk of clashes with other formats. It is recommended that they start with characters that identify the application defining the format name.
9. The data-conversion exit runs in an environment similar to that of the program which issued the MQGET call; environment includes address space and user profile (where applicable). The program could be a message channel agent sending messages to a destination queue manager that does not support message conversion. The exit cannot compromise the queue manager's integrity, since it does not run in the queue manager's environment.
10. The only MQI call which can be used by the exit is MQXCNVC; attempting to use other MQI calls fails with reason code MQRC_CALL_IN_PROGRESS, or other unpredictable errors.
11. No entry point called MQ_DATA_CONV_EXIT is actually provided by the queue manager. However, a **typedef** is provided for the name MQ_DATA_CONV_EXIT in the C programming language, and this can be used to declare the user-written exit, to ensure that the parameters are correct. The name of the exit should be the same as the format name (the name contained in the *Format* field in MQMD), although this is not required in all environments.

The following example illustrates how the exit that processes the format MYFORMAT should be declared in the C programming language:

```
#include "cmqc.h"
#include "cmqxc.h"

MQ_DATA_CONV_EXIT MYFORMAT;

void MQENTRY MYFORMAT(
    PMQDXP  pDataConvExitParms, /* Data-conversion exit parameter
                                block */
    PMQMD   pMsgDesc,           /* Message descriptor */
```

MQ_DATA_CONV_EXIT – Usage notes

```
        MQLONG  InBufferLength,    /* Length in bytes of InBuffer */
        PMQVOID pInBuffer,        /* Buffer containing the unconverted
                                   message */
        MQLONG  OutBufferLength,   /* Length in bytes of OutBuffer */
        PMQVOID pOutBuffer)       /* Buffer containing the converted
                                   message */
    {
        /* C language statements to convert message */
    }
```

12. On z/OS, if an API-crossing exit is also in force, it is called after the data-conversion exit.

C invocation

```
exitname (&DataConvExitParms, &MsgDesc, InBufferLength,
         InBuffer, OutBufferLength, OutBuffer);
```

The parameters passed to the exit are declared as follows:

```
MQDXP  DataConvExitParms; /* Data-conversion exit parameter block */
MQMD   MsgDesc;           /* Message descriptor */
MQLONG InBufferLength;    /* Length in bytes of InBuffer */
MQBYTE InBuffer[n];      /* Buffer containing the unconverted
                           message */
MQLONG OutBufferLength;   /* Length in bytes of OutBuffer */
MQBYTE OutBuffer[n];     /* Buffer containing the converted
                           message */
```

COBOL invocation (OS/400 only)

```
CALL 'exitname' USING DATACONVEXITPARMS, MSGDESC, INBUFFERLENGTH,
                     INBUFFER, OUTBUFFERLENGTH, OUTBUFFER.
```

The parameters passed to the exit are declared as follows:

```
** Data-conversion exit parameter block
01 DATACONVEXITPARMS.
   COPY CMQDXPV.
** Message descriptor
01 MSGDESC.
   COPY CMQMDV.
** Length in bytes of INBUFFER
01 INBUFFERLENGTH PIC S9(9) BINARY.
** Buffer containing the unconverted message
01 INBUFFER PIC X(n).
** Length in bytes of OUTBUFFER
01 OUTBUFFERLENGTH PIC S9(9) BINARY.
** Buffer containing the converted message
01 OUTBUFFER PIC X(n).
```

System/390 assembler invocation

```
CALL EXITNAME,(DATACONVEXITPARMS,MSGDESC,INBUFFERLENGTH,      X
               INBUFFER,OUTBUFFERLENGTH,OUTBUFFER)
```

The parameters passed to the exit are declared as follows:

```
DATACONVEXITPARMS CMQDXPA , Data-conversion exit parameter block
MSGDESC           CMQMDA , Message descriptor
INBUFFERLENGTH   DS      F Length in bytes of INBUFFER
INBUFFER         DS      CL(n) Buffer containing the unconverted
*                                     message
OUTBUFFERLENGTH  DS      F Length in bytes of OUTBUFFER
OUTBUFFER        DS      CL(n) Buffer containing the converted
*                                     message
```

End of product-sensitive programming interface

Object attributes

Appendix G. Signal notification IPC message (Compaq NonStop Kernel only)

MQSeries for Compaq NonStop Kernel, V5.1 supports the signal mode of message-arrival notification. This type of notification is selected by the MQGMO_SET_SIGNAL option in the options field of the Get Message Options structure. If MQGMO_SET_SIGNAL is specified, the following options are not valid:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_NEXT
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_MSG_UNDER_CURSOR
- MQGMO_LOCK
- MQGMO_UNLOCK
- MQGMO_WAIT

If MQGMO_SET_SIGNAL is specified with any of these options, a *CompCode* of MQCC_FAILED and a *Reason* of MQRC_OPTIONS_ERROR are returned.

The effects of specifying MQGMO_SET_SIGNAL are as follows:

- If a message is available when MQGET is issued, it is returned immediately to the requesting application.
- If no message is available when MQGET is issued, a *CompCode* of MQCC_WARNING and a *Reason* of MQRC_SIGNAL_REQUEST_ACCEPTED are returned. When a message becomes available, an Inter-Process Communication (IPC) message is sent to the \$RECEIVE queue of the process that made the MQGET call.

The format of this IPC message is:

MsgCode (INT)

Identifies the message as a notification. The value is TRIGGER_RESPONSE.

ApplTag (LONG)

Is the application tag provided in the *Signal1* field of MQGMO.

The *Signal1* field of MQGMO is significant only when the signal mode of message-arrival notification has been requested. It can be used by an application to associate the IPC notification message with a particular MQGET request.

Status (LONG)

Is the reason Code from MQGET. It can have the following values:

MQRC_NONE

A message satisfying the criteria specified in the MQGET call is available on the queue.

MQRC_NO_MSG_AVAILABLE

The time specified in the *WaitInterval* field has expired.

MQRC_CONNECTION_BROKEN

The queue manager has been stopped.

MQRC_GET_INHIBITED

An operator has inhibited the GET operation for the queue.

Signal notification - Compaq NonStop Kernel

MQRC_Q_DELETED

The queue has been deleted.

MQRC_Q_MGR QUIESCING

The queue manager is quiescing, and the MQGET call was issued with the MQGMO_FAIL_IF QUIESCING option.

MQRC_Q_MGR_STOPPING

The queue manager is shutting down.

Only one signal-notification-mode MQGET call can be outstanding for any queue. If an MQGET with signal notification is specified when there is already a signal-notification MQGET call outstanding for the same queue, a *CompCode* of MQCC_FAILED and a *Reason* of MQRC_SIGNAL_OUSTANDING are returned.

If the signal notification indicates that a message is available (*Status* is MQRC_NONE), the message is not locked by the Queue Manager; therefore, it is also available to any other application that shares the queue. It is possible, therefore, that the message will not be available by the time the application issues an MQGET call to retrieve or browse the message. The signal notification IPC message is not part of any unit of work (that is, a Compaq TMF transaction), started by either the application or MQSeries.

If the application calls MQCLOSE for a queue with outstanding signal-notification MQGET operations initiated by that application, the outstanding signal notifications are cancelled. If an application calls MQDISC, all outstanding signal notifications initiated by the application are cancelled.

Note: Avoid using the GMO WaitInterval with MQGMO_SYNCPOINT. This causes a TMF autoabort of the TMF transaction if no messages are enqueued within the configured autoabort value. An MQRC_UOW_CANCELLED (2297) is returned to the application for the MQGET when a message is enqueued. For more information, see Appendix I of *MQSeries for Compaq NonStop Kernel System Administration*.

Appendix H. Code page conversion

Each national language section lists the following information:

- The native CCSIDs supported
- The code page conversions that are **not** supported

The following terms are used in the information:

-8	Indicates for HP-UX that the CCSID is for the HP-UX defined codeset <i>roman8</i>
AIX	Indicates WebSphere MQ for AIX
COMPAQ-OVMS	Indicates MQSeries for Compaq OpenVMS Alpha
DEC-OVMS	Indicates MQSeries for Digital OpenVMS VAX
HP-UX	Indicates WebSphere MQ for HP-UX
Linux	Indicates WebSphere MQ for Linux for Intel and WebSphere MQ for Linux for zSeries
NCR	Indicates MQSeries for AT&T GIS UNIX
NSK	Indicates MQSeries for Compaq NonStop Kernel
OS/2	Indicates MQSeries for OS/2 Warp
OS/400	Indicates WebSphere MQ for iSeries
SINIX, DC/OSx	Indicates MQSeries for SINIX and DC/OSx
Solaris	Indicates WebSphere MQ for Solaris
Tru64	Indicates MQSeries for Compaq Tru64 UNIX
Windows	Indicates WebSphere MQ for Windows
z/OS	Indicates WebSphere MQ for z/OS

The default for data conversion is for the conversion to be performed at the target (receiving) system.

If the source product supports the conversion a channel can be set up and data exchanged by setting the channel attribute **DataConversion** to YES at the source.

Notes:

1. Conversion for WebSphere MQ client information takes place in the server, so the server must support conversion from the client CCSID to the server CCSID.
2. The conversion may include support added by CSD/PTF to the latest version of WebSphere MQ. Check the content of the latest service level to see if you need to install a CSD/PTF to enable this conversion.

For an extended list of CCSIDs, see the *Character Data Representation Reference*. See Table 101 on page 612 for a cross reference between some of the CCSID numbers and some industry codeset names.

Codeset names and CCSIDs

Table 101. Codeset names and CCSIDs

Codeset names	CCSIDs
ISO 8859-1	819
ISO 8859-2	912
ISO 8859-3	913
ISO 8859-5	915
ISO 8859-6	1089
ISO 8859-7	813
ISO 8859-8	916
ISO 8859-9	920
ISO 8859-13	921
ISO 8859-15 (euro)	923
big5	950
eucJP	954 5050 33722
eucKR	970
eucTW	964
eucCN	1383
PCK	943
GBK	1386
koi8-r	878

WebSphere MQ for z/OS provides more conversion than is listed in the language specific tables. A complete list of conversions provided is shown in Table 102 on page 652.

MQSeries for OS2 Warp provides conversions between CCSIDs in addition to those listed in the language tables. A complete list of conversions provided is shown in "OS/2 conversion support" on page 669.

National languages

The languages supported by WebSphere MQ are:

- US English – see page 614
- German – see page 615
- Danish and Norwegian – see page 616
- Finnish and Swedish – see page 617
- Italian – see page 618
- Spanish – see page 619
- UK English / Gaelic – see page 620
- French – see page 621
- Multilingual – see page 622
- Portuguese – see page 623
- Icelandic – see page 624
- Eastern European languages – see page 625
- Cyrillic – see page 626
- Estonian – see page 627
- Latvian and Lithuanian – see page 628
- Ukranian – see page 629
- Greek – see page 630
- Turkish – see page 631
- Hebrew – see page 632
- Farsi – see page 635
- Urdu – see page 636
- Thai – see page 637
- Lao – see page 638
- Vietnamese – see page 639
- Japanese Latin SBCS – see page 640
- Japanese Katakana SBCS – see page 642
- Japanese Kanji/ Latin Mixed – see page 644
- Japanese Kanji/ Katakana Mixed – see page 646
- Korean – see page 648
- Simplified Chinese – see page 649
- Traditional Chinese – see page 651

US English

US English

The following table shows the native CCSIDs for US English on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	37, 924, 1140
OS/2	437, 858
AIX	819, 850, 5348
HP-UX	819, 923, 1051
NCR	437, 819, 850, 923
Windows	437, 850, 1252, 5348
Compaq-OVMS, DEC-OVMS, NSK, SINIX, DC/OSx, Solaris, Linux	819, 923
Tru64	819, 850, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

OS/400

Code page:

37 Does not convert to code pages 923, 858, 5348

924 Does not convert to code pages 437, 819, 850, 858, 1051, 1140, 1252, 1275, 5348

1140 Does not convert to code pages 924, 1051, 1275, 5348

DEC-OVMS, SINIX, DC/OSx

Code page:

819 Does not convert to code pages 1252, 1275

NCR

Code page:

819 Does not convert to code pages 1252, 1275

437 Does not convert to code pages 1252, 1275

850 Does not convert to code pages 1252, 1275

German

The following table shows the native CCSIDs for German on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	273, 924, 1141
OS/2	850, 858
AIX	819, 850, 5348
HP-UX	819, 923, 1051
NCR	437, 819, 850, 923
Windows	437, 850, 1252, 5348
Compaq-OVMS, DEC-OVMS, NSK, SINIX, DC/OSx, Solaris, Linux, Tru64	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

OS/400

Code page:

273 Does not convert to code pages 858, 923, 924, 1275, 5348

924 Does not convert to code pages 273, 437, 819, 850, 858, 1051, 1141, 1252, 1275, 5348

1141 Does not convert to code pages 924, 1051, 1275, 5348

DEC-OVMS, SINIX, DC/OSx

Code page:

819 Does not convert to code pages 1252, 1275

NCR

Code page:

819 Does not convert to code pages 1252, 1275

437 Does not convert to code pages 1252, 1275

850 Does not convert to code pages 1252, 1275

Danish and Norwegian

Danish and Norwegian

The following table shows the native CCSIDs for Danish and Norwegian on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	277, 924, 1142
OS/2	850, 858, 865
AIX	819, 850, 5348
HP-UX	819, 923, 1051
NCR	819, 850, 865, 923
Windows	850, 865, 1252, 5348
Compaq-OVMS, DEC-OVMS, NSK, SINIX, DC/OSx, Solaris, Linux, Tru64	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

OS/400

Code page:

277 Does not convert to code pages 858, 923, 924, 1275, 5348

924 Does not convert to code pages 277, 819, 850, 858, 865, 1051, 1142, 1252, 1275, 5348

1142 Does not convert to code pages 924, 865, 1051, 1275, 5348

AIX

Code page:

819 Does not convert to code page 865

850 Does not convert to code page 865

HP-UX

Code page:

1051 Does not convert to code page 865

DEC-OVMS, SINIX, DC/OSx

Code page:

819 Does not convert to code pages 1252, 1275

NCR

Code page:

819 Does not convert to code pages 1252, 1275

850 Does not convert to code pages 1252, 1275

865 Does not convert to code pages 1051, 1252, 1275

Windows

Code page:

865 Does not convert to code pages 1051, 1275

Finnish and Swedish

The following table shows the native CCSIDs for Finnish and Swedish on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	278, 924, 1143
OS/2	850, 858, 865
AIX	819, 850, 5348
HP-UX	819, 923, 1051
NCR	437, 819, 850, 923
Windows	437, 850, 865, 1252, 5348
Compaq-OVMS, DEC-OVMS, NSK, SINIX, DC/OSx, Solaris, Linux, Tru64	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

OS/400

Code page:

- 278 Does not convert to code pages 858, 923, 924, 1275, 5348
- 924 Does not convert to code pages 278, 437, 819, 850, 858, 865, 1051, 1143, 1252, 1275, 5348
- 1143 Does not convert to code pages 865, 924, 1051, 1275, 5348

AIX

Code page:

- 819 Does not convert to code page 865
- 850 Does not convert to code page 865

HP-UX

Code page:

- 1051 Does not convert to code page 865

DEC-OVMS, SINIX, DC/OSx

Code page:

- 819 Does not convert to code pages 1252, 1275

NCR

Code page:

- 819 Does not convert to code pages 1252, 1275
- 437 Does not convert to code pages 1252, 1275
- 850 Does not convert to code pages 1252, 1275

Windows

Code page:

- 865 Does not convert to code pages 1051, 1275

Italian

The following table shows the native CCSIDs for Italian on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	280, 924, 1144
OS/2	850, 858
AIX	819, 850, 5348
HP-UX	819, 923, 1051
NCR	437, 819, 850, 923
Windows	437, 850, 1252, 5348
Compaq-OVMS, DEC-OVMS, NSK, SINIX, DC/OSx, Solaris, Linux, Tru64	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

OS/400

Code page:

280 Does not convert to code pages 858, 923, 924, 1275, 5348

924 Does not convert to code pages 280, 437, 819, 850, 858, 1051, 1144, 1252, 1275, 5348

1144 Does not convert to code pages 924, 1051, 1275, 5348

DEC-OVMS, SINIX, DC/OSx

Code page:

819 Does not convert to code pages 1252, 1275

NCR

Code page:

819 Does not convert to code pages 1252, 1275

437 Does not convert to code pages 1252, 1275

850 Does not convert to code pages 1252, 1275

Spanish

The following table shows the native CCSIDs for Spanish on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	284, 924, 1145
OS/2	850, 858
AIX	819, 850, 5348
HP-UX	819, 923, 1051
NCR	437, 819, 850, 923
Windows	437, 850, 1252, 5348
Compaq-OVMS, DEC-OVMS, NSK, SINIX, DC/OSx, Solaris, Linux, Tru64	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

OS/400

Code page:

284 Does not convert to code pages 858, 923, 924, 1275, 5348

924 Does not convert to code pages 284, 437, 819, 850, 858, 1051, 1145, 1252, 1275, 5348

1145 Does not convert to code pages 924, 1051, 1275, 5348

DEC-OVMS, SINIX, DC/OSx

Code page:

819 Does not convert to code pages 1252, 1275

NCR

Code page:

819 Does not convert to code pages 1252, 1275

437 Does not convert to code pages 1252, 1275

850 Does not convert to code pages 1252, 1275

UK English /Gaelic

UK English /Gaelic

The following table shows the native CCSIDs for UK English / Gaelic on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	285, 924, 1146
OS/2	850, 858
AIX	819, 850, 5348
HP-UX	819, 923, 1051
NCR	437, 819, 850, 923
Windows	437, 850, 1252, 5348
Compaq-OVMS, DEC-OVMS, NSK, SINIX, DC/OSx, Solaris, Linux, Tru64	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

OS/400

Code page:

285 Does not convert to code pages 858, 923, 924, 1275, 5348

924 Does not convert to code pages 285, 437, 819, 850, 858, 1051, 1146, 1252, 1275, 5348

1146 Does not convert to code pages 924, 1051, 1275, 5348

DEC-OVMS, SINIX, DC/OSx

Code page:

819 Does not convert to code pages 1252, 1275

NCR

Code page:

819 Does not convert to code pages 1252, 1275

437 Does not convert to code pages 1252, 1275

850 Does not convert to code pages 1252, 1275

French

The following table shows the native CCSIDs for French on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	297, 924, 1147
OS/2	850, 858
AIX	819, 850, 5348
HP-UX	819, 923, 1051
NCR	437, 819, 850, 923
Windows	437, 850, 1252, 5348
Compaq-OVMS, DEC-OVMS, NSK, SINIX, DC/OSx, Solaris, Linux, Tru64	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

OS/400

Code page:

297 Does not convert to code pages 858, 923, 924, 1275, 5348

924 Does not convert to code pages 297, 437, 819, 850, 858, 1051, 1147, 1252, 1275, 5348

1147 Does not convert to code pages 924, 1051, 1275, 5348

DEC-OVMS, SINIX, DC/OSx

Code page:

819 Does not convert to code pages 1252, 1275

NCR

Code page:

819 Does not convert to code pages 1252, 1275

437 Does not convert to code pages 1252, 1275

850 Does not convert to code pages 1252, 1275

Multilingual

Multilingual

The following table shows the native CCSIDs for multilingual conversion on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	500, 924, 1148
OS/2	850, 858
AIX	819, 850, 5348
HP-UX	819, 923, 1051
NCR	437, 819, 850, 923
Windows	437, 850, 1252, 5348
Compaq-OVMS, DEC-OVMS, NSK, SINIX, DC/OSx, Solaris, Linux, Tru64	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

OS/400

Code page:

500 Does not convert to code pages 858, 923, 5348

924 Does not convert to code pages 437, 819, 850, 858, 1051, 1148, 1252, 1275, 5348

1148 Does not convert to code pages 924, 1051, 1275, 5348

DEC-OVMS, SINIX, DC/OSx

Code page:

819 Does not convert to code pages 1252, 1275

NCR

Code page:

819 Does not convert to code pages 1252, 1275

437 Does not convert to code pages 1252, 1275

850 Does not convert to code pages 1252, 1275

Portuguese

The following table shows the native CCSIDs for Portuguese on supported platforms:

Platform	Native CCSIDs
OS/400	37, 500, 924, 1140
z/OS	500, 924, 1140
OS/2	850, 858, 860
AIX	819, 850, 5348
HP-UX	819, 923, 1051
NCR	819, 850, 860, 923
Windows	850, 860, 1252, 5348
Compaq-OVMS, DEC-OVMS, NSK, SINIX, DC/OSx, Solaris, Linux, Tru64	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

OS/400

Code page:

- 37 Does not convert to code pages 858, 923, 1275, 5348
- 500 Does not convert to code pages 858, 923, 1275, 5348
- 924 Does not convert to code pages 819, 850, 858, 860, 1051, 1140, 1252, 1275, 5348
- 1140 Does not convert to code pages 860, 924, 1051, 1275, 5348

HP-UX

Code page:

- 1051 Does not convert to code page 860

DEC-OVMS, SINIX, DC/OSx

Code page:

- 819 Does not convert to code pages 1252, 1275

NCR

Code page:

- 819 Does not convert to code pages 1252, 1275
- 850 Does not convert to code pages 1252, 1275
- 860 Does not convert to code pages 1051, 1252, 1275

Windows

Code page:

- 860 Does not convert to code pages 1051, 1275

Icelandic

Icelandic

The following table shows the native CCSIDs for Icelandic on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	871, 924, 1149
OS/2	850, 858, 861
AIX	819, 850, 5348
HP-UX	819, 923, 1051
NCR	819, 850, 923
Windows	850, 861, 1252, 5348
Compaq-OVMS, DEC-OVMS, NSK, SINIX, DC/OSx, Solaris, Linux, Tru64	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

OS/400

Code page:

871 Does not convert to code pages 858, 923, 924, 1275, 5348

924 Does not convert to code pages 819, 850, 858, 861, 871, 1051, 1149, 1252, 1275, 5348

1149 Does not convert to code pages 924, 1051, 1275, 5348

HP-UX

Code page:

1051 Does not convert to code page 861

DEC-OVMS, SINIX, DC/OSx

Code page:

819 Does not convert to code pages 1252, 1275

NCR

Code page:

819 Does not convert to code pages 1252, 1275

850 Does not convert to code pages 1252, 1275

Windows

Code page:

861 Does not convert to code pages 1051, 1275

Eastern European languages

The typical languages using these CCSIDs include Albanian, Croatian, Czech, Hungarian, Polish, Romanian, Serbian, Slovakian, and Sloven.

The following table shows the native CCSIDs for Eastern European languages on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	870
OS/2	852
Windows	852, 1250, 5346
AIX, Compaq-OVMS, DEC-OVMS, HP-UX, NCR, NSK, SINIX, DC/OSx, Solaris, Linux, Tru64	912
Eastern European Apple client	1282
Romanian Apple client	1285
Croatian Apple client	1284

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

z/OS

Code page:

870 Does not convert to code pages 1284, 1285

OS/400

Code page:

870 Does not convert to code pages 1284, 1285, 5346

HP-UX, Solaris, Linux, DEC-OVMS, Compaq-OVMS, SINIX, DC/OSx

Code page:

912 Does not convert to code pages 1284, 1285

NCR

Code page:

912 Does not convert to code pages 1250, 1282, 1284, 1285, 5346

Windows

Code page:

852 Does not convert to code pages 1284, 1285

1250 Does not convert to code pages 1284, 1285

Cyrillic

Cyrillic

The typical languages using these CCSIDs include Byelorussia (Belarus), Bulgarian, Macedonian, Russian, and Serbian.

The following table shows the native CCSIDs for Cyrillic on supported platforms:

Platform	Native CCSIDs
z/OS	1025
OS/400	880, 1025
OS/2	855, 866, 1131
Windows	855, 866, 1131, 1251, 5347
Solaris	878, 915
AIX, Compaq-OVMS, DEC-OVMS, HP-UX, NCR, NSK, SINIX, DC/OSx, Tru64	915
Apple client	1283

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

OS/400

Code page:

880 Does not convert to code pages 855, 866, 878, 1131, 5347

1025 Does not convert to code pages 878, 5347

NCR

Code page:

915 Does not convert to code pages 878, 1131, 1251, 1283

DEC-OVMS, SINIX, DC/OSx

Code page:

915 Does not convert to code pages 878, 1131

Windows

Code page:

855 Does not convert to code page 1131

866 Does not convert to code page 1131

1131 Does not convert to code pages 855, 866, 880, 1283

Estonian

The following table shows the native CCSIDs for Estonian on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	1122
Windows	922, 1257, 5353
AIX, Compaq-OVMS, DEC-OVMS, HP-UX, NSK, OS/2, Solaris, Linux, Tru64	922
NCR, SINIX, DC/OSx	Not known

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

OS/400

Code page:

1122 Does not convert to code page 5353

DEC-OVMS

Code page:

922 Does not convert to code pages 1122, 1257, 5353

Latvian and Lithuanian

Latvian and Lithuanian

The following table shows the native CCSIDs for Latvian and Lithuanian on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	1112
Windows	921, 1257, 5353
AIX, Compaq-OVMS, DEC-OVMS, HP-UX, NSK, OS/2 Solaris, Linux, Tru64	921
NCR, SINIX, DC/OSx	Not known

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

OS/400

Code page:

1112 Does not convert to code page 5353

DEC-OVMS

Code page:

921 Does not convert to code pages 1112, 1257, 5353

Ukrainian

The following table shows the native CCSIDs for Ukrainian on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	1123
OS/2	1125
Windows	1124, 1125, 1251, 5347
AIX, Compaq-OVMS, DEC-OVMS, HP-UX, NSK, Solaris, Linux, Tru64	1124
NCR, SINIX, DC/OSx	Not known

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

OS/400

Code page:

1123 Does not convert to code page 5347

HP-UX

Code page:

1124 Does not convert to code page 5347

DEC-OVMS

Code page:

1124 Does not convert to code pages 1123, 1125, 1251, 5347

Windows

Code page:

1125 Does not convert to code page 1123

Greek

Greek

The following table shows the native CCSIDs for Greek on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	875
OS/2	813, 869
HP-UX	813 (see note)
Windows	869, 1253, 5349
AIX, Compaq-OVMS, DEC-OVMS, NCR, NSK, SINIX, DC/OSx, Solaris, Linux, Tru64	813
Apple client	1280
DOS client	737

Note: Only the ISO codeset is supported on HP-UX. The HP-UX proprietary greek8 codeset has no registered CCSID and is not supported.

All non-client platforms support conversion between their native CCSIDs, the native CCSIDs of the other platforms with the following exceptions.

OS/400

Code page:

875 Does not convert to code page 5349

NCR

Code page:

813 Does not convert to code pages 737, 1253, 1280

DEC-OVMS, SINIX, DC/OSx

Code page:

813 Does not convert to code page 737

Windows

Code page:

1253 Does not convert to code page 737

5349 Does not convert to code page 737

Turkish

The following table shows the native CCSIDs for Turkish on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	1026
OS/2	857
HP-UX	920 (see note)
Windows	857, 1254, 5350
NCR	Not known
AIX, Compaq-OVMS, DEC-OVMS, NSK, SINIX, DC/OSx, Solaris, Linux, Tru64	920
Apple client	1281
Note: Only the ISO codeset is supported on HP-UX. The HP-UX proprietary turkish8 codeset has no registered CCSID and is not supported.	

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

OS/400

Code page:

1026 Does not convert to code page 5350

Hebrew

The following table shows the native CCSIDs for Hebrew on supported platforms:

Platform	Native CCSIDs
z/OS	424, 803, 4899, 12712
OS/400	424
OS/2	862, 867
AIX	856, 916, 9048
HP-UX	916 (see note)
Windows	1255, 5351
Compaq-OVMS, DEC-OVMS, NSK, SINIX, DC/OSx, Solaris, Linux, Tru64	916
NCR	Not known
Note: Only the ISO codeset is supported on HP-UX. The HP-UX proprietary greek8 codeset has no registered CCSID and is not supported.	

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS

Code page:

- 424 Does not convert to code pages 867, 4899, 9048, 12712
- 803 Does not convert to code pages 867, 4899, 5351, 9048, 12712
- 4899 Does not convert to code pages 424, 803, 856, 862, 916, 1255
- 12712 Does not convert to code pages 424, 803, 856, 916, 1255

OS/400

Code page:

- 424 Does not convert to code pages 803, 867, 4899, 5351, 9048, 12712
Code page 424 also converts to and from CCSID 4952, which is a variant of 856.

AIX

Code page:

- 856 Does not convert to code pages 867, 4899, 9048, 12712
- 916 Does not convert to code pages 867, 4899, 9048, 12712
- 9048 Does not convert to code pages 424, 803, 856, 862, 916, 1255

Compaq-OVMS, NSK, Tru64, HP-UX, Solaris, Linux

Code page:

- 916 Does not convert to code pages 867, 4899, 9048, 12712

SINIX, DC/OSx

Code page:

- 916 Does not convert to code pages 803, 916, 4899, 9048, 12712

DEC-OVMS

Code page:

916 Does not convert to code pages 803, 867, 4899, 9048, 12712

Windows

Code page:

1255 Does not convert to code pages 867, 4899, 9048, 12712

5351 Does not convert to code page 803

Arabic

The following table shows the native CCSIDs for Arabic on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	420
OS/2	864
AIX	1046, 1089
HP-UX	1089 (see note)
Windows	720, 864, 1256, 5352
NCR	Not known
Compaq-OVMS, DEC-OVMS, NSK, SINIX, DC/OSx, Solaris, Linux, Tru64	1089
Note: Only the ISO codeset is supported on HP-UX. The HP-UX proprietary arabic8 codeset has no registered CCSID and is not supported.	

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

OS/400

Code page:

420 Does not convert to code page 5352

OS/2

Code page:

864 Does not convert to code page 720

HP-UX, Solaris, Linux, Compaq-OVMS, DEC-OVMS, SINIX, DC/OSx, NSK, Tru64

Code page:

1089 Does not convert to code page 720

Windows

Code page:

720 Does not convert to code pages 1089, 5352

5352 Does not convert to code page 720

Farsi

The following table shows the native CCSIDs for Farsi on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	1097
OS/2	1098
NCR, SINIX, DC/OSx	Not known
AIX, Compaq-OVMS, DEC-OVMS, HP-UX, NSK, Solaris, Linux, Tru64, Windows	1098 (see note)
Note: The native CCSID for these platforms has not been standardized and may change.	

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

DEC-OVMS

Code page:

1098 Does not convert to code page 1097

Urdu

Urdu

The following table shows the native CCSIDs for Urdu on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	918
OS/2, Windows	868
NCR, SINIX, DC/OSx	Not known
AIX, HP-UX, Compaq-OVMS, DEC-OVMS, NSK, Solaris, Linux, Tru64	1006

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

OS/400

Code page:

918 Does not convert to code page 1006

DEC-OVMS

Code page:

1006 Does not convert to code pages 868, 918

Thai

The following table shows the native CCSIDs for Thai on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	838
OS/2	874
NCR, SINIX, DC/OSx	Not known
AIX, Compaq-OVMS, DEC-OVMS, HP-UX, NSK, Solaris, Linux, Tru64, Windows	874 (see note)
Note: The native CCSID for these platforms has not been standardized and may change.	

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

DEC-OVMS

Code page:

874 Does not convert to code page 838

Lao

Lao

The following table shows the native CCSIDs for Lao on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	1132
NCR, SINIX, DC/OSx	Not known
AIX, Compaq-OVMS, DEC-OVMS, HP-UX, OS/2, NSK, Solaris, Linux, Tru64, Windows	1133

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

DEC-OVMS

Code page:

1133 Does not convert to code page 1132

Vietnamese

The following table shows the native CCSIDs for Vietnamese on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	1130
Windows	1258, 5354
NCR, SINIX, DC/OSx	Not known
AIX, Compaq-OVMS, DEC-OVMS, HP-UX, NSK, OS/2, Solaris, Linux, Tru64	1129

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

OS/400

Code page:

1130 Does not convert to code pages 1129, 5354

DEC-OVMS

Code page:

1129 Does not convert to code pages 1130, 1258, 5354

Japanese Latin SBCS

Japanese Latin SBCS

The following table shows the native CCSIDs for Japanese Latin SBCS on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	1027
OS/2	932, 942
AIX	932, 5050, 33722 (see Note 1)
Windows	932, 943 (see Note 2)
Solaris	943, 5050
HP-UX, NCR, SINIX, DC/OSx	Not known
DEC-OVMS	943, 954
Compaq-OVMS	943, 5050
NSK	943, 5050
Tru64	943, 954, 5050, 33722

Notes:

1. 5050 and 33722 are CCSIDs related to base code page 954 on AIX. The CCSID reported by the operating system is 33722.
2. Windows NT uses code page 932 but this is best represented by the CCSID of 943. However, not all platforms of WebSphere MQ support this CCSID.
On WebSphere MQ for Windows CCSID 932 is used to represent code page 932, but a change to file `../conv/table/ccsid.tbl` can be made which changes the CCSID used to 943.

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS

Code page:

1027 Does not convert to code pages 932, 942, 943, 954, 5050, 33722

OS/400

Code page:

1027 Does not convert to code page 932

AIX

Code page:

932 Does not convert to code page 1027

5050 Does not convert to code page 1027

33722 Does not convert to code page 1027

Solaris

Code page:

943 Does not convert to code page 1027

5050 Does not convert to code page 1027

DEC-OVMS

Code page:

Japanese Latin SBCS

943 Does not convert to code pages 932, 942, 954, 1027, 5050, 33722

954 Does not convert to code pages 932, 942, 943, 1027

Compaq-OVMS, NSK

Code page:

943 Does not convert to code page 1027

5050 Does not convert to code page 1027

Tru64

Code page:

943 Does not convert to code page 1027

954 Does not convert to code page 1027

5050 Does not convert to code page 1027

33722 Does not convert to code page 1027

Japanese Katakana SBCS

Japanese Katakana SBCS

The following table shows the native CCSIDs for Japanese Katakana SBCS on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	290
OS/2, HP-UX	897
AIX	932, 5050, 33722 (see Note 1)
Windows	932, 943 (see Note 2)
Solaris	943, 5050
NCR, SINIX, DC/OSx	Not known
DEC-OVMS	943, 954
Compaq-OVMS	943, 5050
NSK	943, 5050
Tru64	943, 954, 5050, 33722

Notes:

1. 5050 and 33722 are CCSIDs related to base code page 954 on AIX. The CCSID reported by the operating system is 33722.
2. Windows NT uses code page 932 but this is best represented by the CCSID of 943. However, not all platforms of WebSphere MQ support this CCSID.
On WebSphere MQ for Windows CCSID 932 is used to represent code page 932, but a change to file `../conv/table/ccsid.tbl` can be made which changes the CCSID used to 943.
3. In addition to the above conversions, the WebSphere MQ products on AIX, OS/2, HP-UX, Solaris, Linux and Tru64 support conversion from CCSID 897 to CCSIDs 37, 273, 277, 278, 280, 284, 285, 290, 297, 437, 500, 819, 850, 1027, and 1252.

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS

Code page:

290 Does not convert to code pages 932, 943, 954, 5050, 33722

OS/400

Code page:

290 Does not convert to code page 932

AIX

Code page:

932 Does not convert to code pages 290, 897

5050 Does not convert to code pages 290, 897

33722 Does not convert to code pages 290, 897

HP-UX

Code page:

897 Does not convert to code pages 932, 943, 954, 5050, 33722

Solaris

Code page:

943 Does not convert to code pages 290, 897

5050 Does not convert to code pages 290, 897

DEC-OVMS

Code page:

943 Does not convert to code pages 290, 897, 932, 954, 5050, 33722

954 Does not convert to code pages 290, 897, 943

Compaq-OVMS, NSK

Code page:

943 Does not convert to code pages 290, 897

5050 Does not convert to code pages 290, 897

Tru64

Code page:

943 Does not convert to code pages 290, 897

954 Does not convert to code pages 290, 897

5050 Does not convert to code pages 290, 897

33722 Does not convert to code pages 290, 897

Japanese Kanji/ Latin Mixed

Japanese Kanji/ Latin Mixed

The following table shows the native CCSIDs for Japanese Kanji/ Latin Mixed on supported platforms:

Platform	Native CCSIDs
OS/400, z/OS	1399, 5035 (see Note 1)
OS/2	932, 942
AIX	932, 5050, 33722 (see Note 2)
HP-UX	932, 954, 5039 (see Note 3)
Windows	932, 943 (see Note 4)
Solaris	943, 5050
NCR, SINIX, DC/OSx	Not known
Compaq-OVMS	943, 5050
DEC-OVMS	932, 954
NSK	943, 5050
Tru64	943, 954, 5050, 33722

Notes:

- 5035 is a CCSID related to code page 939
- 5050 and 33722 are CCSIDs related to base code page 954 on AIX. The CCSID reported by the operating system is 33722.
- Code sets japan15 and SJIS on HP-UX are represented by CCSID 932. These have a few DBCS characters having different representations in SJIS so 932 may be converted incorrectly if the conversion is not performed on an HP-UX system. WebSphere MQ for HP-UX supports 5039, the correct CCSID for HP SJIS. A change to file `/var/mqm/conv/ccsid.tbl` can be made to change the CCSID used from 932 to 5039.
- Windows NT uses code page 932 but this is best represented by the CCSID of 943. However, not all platforms of WebSphere MQ support this CCSID.
On WebSphere MQ for Windows CCSID 932 is used to represent code page 932, but a change to file `../conv/table/ccsid.tbl` can be made which changes the CCSID used to 943.

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS

Code page:

1399 Does not convert to code pages 954, 5035, 5050, 33722

5035 Does not convert to code pages 954, 1399, 5050, 33722

OS/400

Code page:

1399 Does not convert to code page 5039

5035 Does not convert to code page 5039

HP-UX

Code page:

932 Does not convert to code pages 942, 943, 1399

954 Does not convert to code pages 942, 943, 1399

5039 Does not convert to code pages 942, 943, 1399

DEC-OVMS

Code page:

932 Does not convert to code pages 942, 943, 1399, 5039

954 Does not convert to code pages 942, 943, 1399, 5039

Compaq-OVMS, NSK

Code page:

943 Does not convert to code page 1399

5050 Does not convert to code page 1399

Tru64

Code page:

943 Does not convert to code page 1399

954 Does not convert to code page 1399

5050 Does not convert to code page 1399

33722 Does not convert to code page 1399

Japanese Kanji/ Katakana Mixed

Japanese Kanji/ Katakana Mixed

The following table shows the native CCSIDs for Japanese Kanji/ Katakana Mixed on supported platforms:

Platform	Native CCSIDs
z/OS	1390, 5026 (see Note 1)
OS/400	5026 (see Note 1)
OS/2	932, 942
AIX	932, 5050, 33722 (see Note 2)
HP-UX	932, 954, 5039 (see Note 3)
Windows	932, 943 (see Note 4)
Solaris	943, 5050
NCR, SINIX, DC/OSx	Not known
Compaq-OVMS	943, 5050
DEC-OVMS	932, 954
NSK	943, 5050
Tru64	943, 954, 5050, 33722

Notes:

1. 5026 is a CCSID related to code page 930. CCSID 5026 is the CCSID reported on OS/400 when the Japanese Katakana (DBCS) feature is selected.
2. 5050 and 33722 are CCSIDs related to base code page 954 on AIX. The CCSID reported by the operating system is 33722.
3. Code sets japan15 and SJIS on HP-UX are represented by CCSID 932. These have a few DBCS characters having different representations in SJIS so 932 may be converted incorrectly if the conversion is not performed on an HP-UX system. WebSphere MQ for HP-UX supports 5039, the correct CCSID for HP SJIS. A change to file `/var/mqm/conv/ccsid.tbl` can be made to change the CCSID used from 932 to 5039.
4. Windows NT uses code page 932 but this is best represented by the CCSID of 943. However, not all platforms of WebSphere MQ support this CCSID. On WebSphere MQ for Windows, CCSID 932 is used to represent code page 932, but a change to file `./conv/table/ccsid.tbl` can be made that changes the CCSID used to 943.

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS

Code page:

1390 Does not convert to code pages 954, 5026, 5050, 33722

5026 Does not convert to code pages 954, 1390, 5050, 33722

OS/400

Code page:

5026 Does not convert to code pages 1390, 5039

HP-UX

Code page:

932 Does not convert to code pages 942, 943, 1390

Japanese Kanji/ Katakana Mixed

954 Does not convert to code pages 942, 943, 1390

5039 Does not convert to code pages 942, 943, 1390

DEC-OVMS

Code page:

932 Does not convert to code pages 942, 943, 1390, 5039

954 Does not convert to code pages 942, 943, 1390, 5039

Compaq-OVMS, NSK

Code page:

943 Does not convert to code page 1390

5050 Does not convert to code page 1390

Tru64

Code page:

943 Does not convert to code page 1390

954 Does not convert to code page 1390

5050 Does not convert to code page 1390

33722 Does not convert to code page 1390

Korean

Korean

The following table shows the native CCSIDs for Korean on supported platforms:

Platform	Native CCSIDs
z/OS, OS/400	933, 1364
OS/2	949, 1363
AIX, HP-UX	970
Windows	949, 1363
Solaris	970
NCR, SINIX, DC/OSx	Not known
Compaq-OVMS , DEC-OVMS, NSK, Tru64	970

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS

Code page:

933 Does not convert to code page 970

1364 Does not convert to code page 970

HP-UX

Code page:

970 Does not convert to code pages 949, 1363, 1364

DEC-OVMS

Code page:

970 Does not convert to code pages 1363, 1364

Simplified Chinese

The following table shows the native CCSIDs for Simplified Chinese on supported platforms:

Platform	Native CCSIDs
z/OS	935, 1388
OS/400	935
OS/2	1381, 1386
AIX	1383, 1386
HP-UX	1381 (see Note 1)
Windows	1381, 1386(see Note 2)
Solaris	1383
NCR, SINIX, DC/OSx	Not known
Compaq-OVMS, DEC-OVMS, NSK, Tru64	1383

Notes:

- Code sets prc15 and hp15CN on HP-UX are represented by CCSID 1381.
- Windows NT uses code page 936 but this is best represented by the CCSID of 1386. However, not all platforms of WebSphere MQ support this CCSID.
On WebSphere MQ for Windows CCSID 1381 is used to represent code page 936, but a change to file ../conv/table/ccsid.tbl can be made which changes the CCSID used to 1386.
- WebSphere MQ V5.3 supports phase one of the Chinese GB18030 standard.
On z/OS, Linux, Windows, and Solaris, conversion support is provided between Unicode (UTF-8 and UCS-2) and CCSID 1388 (EBCDIC with GB18030 extensions), Unicode (UTF-8 and UCS-2) and CCSID 5488 (GB18030 phase one), and between CCSID 1388 and CCSID 5488.
Note: For MQSeries on AIX V5.1 only, the appropriate GB18030 PTF must be installed.
On AIX V4.3.3 there is no operating system support for GB18030. On AIX V5.1, APAR IY26937 provides support for conversion between GB18030 (CCSID 5488) and Unicode.
On OS/400, support is provided by the operating system for conversion between Unicode (UTF-8 and UCS-2) and CCSID 1388 (EBCDIC with GB18030 extensions).
On HP-UX there is currently no support available on the HP11 operating system for GB18030. On HP11i, patch PHCO_26456 provides conversion support between GB18030 (CCSID 5488) and Unicode. Support is not provided for the conversion between GB18030 and 1388 (EBCDIC).
Refer to the Readme file provided with the product for the latest information on GB18030 support.

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS

Code page:

- 935 Does not convert to code page 1383
- 1388 Does not convert to code page 1383

Simplified Chinese

HP-UX

Code page:

1381 Does not convert to code pages 1383, 1386, 1388

DEC-OVMS

Code page:

1383 Does not convert to code pages 1386, 1388

Traditional Chinese

The following table shows the native CCSIDs for Traditional Chinese on supported platforms:

Platform	Native CCSIDs
z/OS, OS/400	937
OS/2	938, 948, 950
HP-UX	938, 950, 964 (see Note)
Tru64, Windows	950
NCR, SINIX, DC/OSx	Not known
AIX, Compaq-OVMS, DEC-OVMS, NSK, Solaris, Linux	950, 964
Note: Code set roc15 on HP-UX is represented by CCSID 938.	

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS

Code page:

937 Does not convert to code page 964

1388 Does not convert to code page 1383

HP-UX

Code page:

938 Does not convert to code page 948

950 Does not convert to code page 948

964 Does not convert to code page 948

Compaq-OVMS/Solaris

Code page:

964 Does not convert to code page 938

z/OS conversion support

Table 102. WebSphere MQ for z/OS CCSID conversion support

CCSID	Converts to and from CCSIDS
37	256, 273, 275, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 720, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-866, 869-871, 874-875, 880, 897, 903-905, 912, 914-916, 920-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1097, 1100, 1112, 1114-1115, 1122, 1124, 1126, 1130-1132, 1137, 1140-1149, 1200, 1208, 1250-1255, 1257-1258, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5210-5211, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 16804, 17248, 17584, 25473, 25479, 25480, 25617, 25619, 25664, 28709
256	37, 273, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 737, 775, 819, 833, 836, 838, 850, 852, 857, 860-866, 869-871, 875, 880, 905, 1025-1027, 1112, 1122, 1200, 1208, 1251-1252, 1275, 4386, 4929, 4932, 4934, 4946, 4948, 4953, 4960, 4971, 5123, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 13121, 13488, 16804, 17248, 17584, 28709
259	437, 808, 850-852, 855-858, 860-865, 867, 869, 872, 874, 899, 901-902, 915, 1098, 1161-1162, 1200, 1208, 1250-1258, 4946, 4948, 4951-4953, 4960, 4970, 5346, 5348, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584
273	37, 256, 277-278, 280, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1250, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951-4953, 4960, 4970-4971, 5012, 5123, 5346, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
274	500, 1047
275	37, 437, 500, 819, 850, 1047, 1200, 1208, 1252, 4946, 5348, 8229, 13488, 17584, 28709
277	37, 256, 273, 278, 280, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
278	37, 256, 273, 277, 280, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
280	37, 256, 273, 277-278, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
281	1047
282	500, 1047, 1200, 1208, 13488, 17584
284	37, 256, 273, 277-278, 280, 285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709

Table 102. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
285	37, 256, 273, 277-278, 280, 284, 290, 297, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
290	37, 256, 273, 277-278, 280, 284-285, 297, 367, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 860-865, 870-871, 895-897, 1009, 1025-1027, 1040-1043, 1088, 1112, 1122, 1139, 1200, 1208, 1252, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4960, 4992, 5123, 8229, 8482, 9025, 9044, 9049, 9056, 13121, 13488, 17248, 17584, 25473, 25617, 25619, 25664, 28709
293	1200, 1208, 13488, 17584
297	37, 256, 273, 277-278, 280, 284-285, 290, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
300	301, 941, 1200, 1208, 1351, 4396, 8492, 13488, 16684, 17584
301	300, 941, 1200, 1208, 1351, 4396, 8492, 13488, 16684, 17584
367	37, 256, 273, 277-278, 280, 284, 290, 297, 500, 819, 833, 836, 850, 871, 875, 1009, 1026-1027, 1041, 1088, 1115, 1126, 1200, 1208, 4386, 4929, 4932, 4946, 4971, 5123, 5211, 8229, 8482, 9025, 13121, 13488, 17584, 25617, 25664, 28709
420	37, 256, 424, 437, 500, 720, 737, 775, 819, 850, 852, 857, 860-865, 1008, 1046, 1089, 1098, 1112, 1122, 1127, 1200, 1208, 1252, 1256, 4946, 4948, 4953, 4960, 5104, 5142, 5352, 8229, 8612, 9044, 9049, 9056, 9238, 13488, 16804, 17248, 17584, 28709
423	37, 256, 273, 277-278, 280, 284-285, 297, 437, 500, 737, 775, 813, 819, 838, 850-852, 857, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1009, 1025-1027, 1041-1043, 1112, 1122, 1200, 1208, 1252-1253, 1280, 4909, 4934, 4946, 4948, 4953, 4960, 4970-4971, 5012, 5123, 8229, 9030, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 28709
424	37, 256, 420, 437, 500, 737, 775, 803, 819, 836, 850, 852, 856-857, 860-865, 916, 1112, 1122, 1200, 1208, 1252, 1255, 4932, 4946, 4948, 4952-4953, 4960, 5012, 5351, 8229, 8612, 9044, 9049, 9056, 13488, 16804, 17248, 17584, 28709
437	37, 256, 259, 273, 275, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-863, 865-866, 869-871, 874-875, 880, 897, 903, 905, 912, 914-916, 920-924, 1025-1027, 1040-1043, 1047, 1051, 1097, 1098, 1114-1115, 1126, 1140-1149, 1200, 1208, 1252, 1257, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5210-5211, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 28709
500	37, 256, 273-275, 277-278, 280, 282, 284-285, 290, 297, 367, 420, 423-424, 437, 737, 775, 813, 819, 833, 836, 838, 850-852, 855-858, 860-866, 869-871, 874-875, 880, 891, 895, 897, 903-905, 912, 914-916, 920-924, 1004, 1009-1021, 1023, 1025-1027, 1040-1043, 1046-1047, 1051, 1088-1089, 1097, 1100-1107, 1112, 1114-1115, 1122, 1124-1126, 1129-1133, 1137, 1140-1149, 1200, 1208, 1250-1258, 1275, 1280-1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951-4953, 4960, 4970-4971, 5012, 5123, 5142, 5210-5211, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 9238, 13121, 13488, 16804, 17248, 17584, 25473, 25479, 25480, 25617, 25619, 25664, 28709
720	37, 420, 864, 1200, 1208, 1256, 4960, 8229, 8612, 9056, 13488, 16804, 17248, 17584, 28709

z/OS conversion support

Table 102. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
737	37, 256, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 833, 836, 838, 850, 869-871, 875, 880, 905, 1025-1027, 1097, 1200, 1208, 1252-1253, 1280, 4386, 4909, 4929, 4932, 4934, 4946, 4971, 5123, 8229, 8482, 8612, 9025, 9030, 9061, 13121, 13488, 16804, 17584, 28709
775	37, 256, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 833, 836, 838, 850, 870-871, 875, 880, 905, 1025-1027, 1097, 1112, 1122, 1200, 1208, 1252, 1257, 4386, 4929, 4932, 4934, 4946, 4971, 5123, 8229, 8482, 8612, 9025, 9030, 13121, 13488, 16804, 17584, 28709
803	424, 819, 850, 856, 862, 916, 1200, 1208, 1252, 1255, 4946, 4952, 5012, 13488, 17584
806	1200, 1208, 13488, 17584
808	259, 858-859, 872, 923-924, 1140, 1148, 1153-1154, 1200, 1208, 5347, 5348, 13488, 17584
813	37, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252-1253, 1280, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5349, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
819	37, 256, 273, 275, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 803, 813, 833, 836, 838, 850, 852, 855, 857-858, 860-861, 863-866, 869-871, 874-875, 880, 897, 903, 905, 912, 914-916, 920-924, 1004, 1025-1027, 1041-1043, 1047, 1051, 1088-1089, 1097, 1098, 1112, 1114, 1122-1123, 1126, 1130, 1132, 1137, 1140-1149, 1200, 1208, 1250-1255, 1257-1258, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5210, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 16804, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
833	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 437, 500, 737, 775, 819, 836, 850, 852, 855, 857, 860-865, 870-871, 891, 1009, 1025-1027, 1040-1043, 1088, 1112, 1122, 1126, 1200, 1208, 1252, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4960, 5123, 8229, 8482, 9025, 9044, 9049, 9056, 13121, 13488, 17248, 17584, 25617, 25619, 25664, 28709
834	926, 951, 1200, 1208, 1362, 4930, 9026, 13488, 17584
835	927, 947, 1200, 1208, 4931, 9027, 13488, 17584, 21427
836	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 424, 437, 500, 737, 775, 819, 833, 850, 852, 855, 857, 870-871, 875, 903, 1009, 1025-1027, 1040-1043, 1088, 1112, 1114-1115, 1122, 1200, 1208, 1252, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4971, 5123, 5210-5211, 8229, 8482, 9025, 9044, 9049, 13121, 13488, 17584, 25479, 25617, 25619, 25664, 28709
837	928, 1200, 1208, 1380, 1385, 4933, 13488, 17584
838	37, 256, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 775, 813, 819, 850, 852, 857, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1112, 1122, 1200, 1208, 1252, 4909, 4934, 4946, 4948, 4953, 4960, 4970-4971, 5012, 5123, 8229, 9030, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 28709
848	924, 1148, 1158, 1200, 1208, 5347, 13488, 17584
849	924, 1148, 1154, 1200, 1208, 5347, 13488, 17584
850	37, 256, 259, 273, 275, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 737, 775, 803, 813, 819, 833, 836, 838, 852, 855-858, 860-866, 869-871, 874-875, 880, 897, 903, 905, 912, 914-916, 920-924, 1004, 1025-1027, 1040-1043, 1047, 1051, 1088-1089, 1097, 1098, 1100, 1112, 1114, 1122, 1126, 1130, 1132, 1140-1149, 1200, 1208, 1250-1257, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951-4953, 4960, 4970-4971, 5012, 5123, 5210, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 16804, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
851	259, 423, 500, 875, 1200, 1208, 4971, 13488, 17584

Table 102. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
852	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 836, 838, 850, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1040-1043, 1047, 1088, 1097, 1200, 1208, 1250, 1252, 1282, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5346, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 25664, 28709
855	37, 259, 273, 277-278, 280, 284-285, 290, 297, 437, 500, 819, 833, 836, 850, 852, 857, 866, 870-871, 878, 880, 912, 915, 1025-1027, 1040-1043, 1088, 1200, 1208, 1250-1252, 1283, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 5123, 5346, 5347, 8229, 8482, 9025, 9044, 9049, 13121, 13488, 17584, 25617, 25619, 25664, 28709
856	259, 273, 424, 500, 803, 850, 862, 916, 1200, 1208, 1255, 4946, 4952, 5012, 5351, 13488, 17584
857	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1040-1043, 1088, 1097, 1200, 1208, 1252, 1254, 1281, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5350, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 25664, 28709
858	37, 259, 273, 277-278, 280, 284-285, 297, 437, 500, 808, 819, 850, 860-861, 865, 871-872, 901-902, 923-924, 1047, 1051, 1140-1149, 1153-1157, 1160-1162, 1164, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
859	808, 872, 901-902, 1153-1157, 1160-1162, 1164, 1200, 1208, 13488, 17584
860	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 838, 850, 852, 857-858, 861, 863, 865, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 923-924, 1025-1027, 1041-1043, 1097, 1140, 1145-1146, 1148, 1200, 1208, 1252, 4386, 4909, 4929, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 28709
861	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 838, 850, 852, 857-858, 860, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 923-924, 1025-1027, 1041-1043, 1097, 1148, 1149, 1200, 1208, 1252, 4386, 4909, 4929, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 28709
862	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 803, 833, 838, 850, 856, 870-871, 875, 880, 905, 916, 1025-1027, 1097, 1200, 1208, 1252, 1255, 4386, 4929, 4934, 4946, 4952, 4971, 5012, 5123, 5351, 8229, 8482, 8612, 9025, 9030, 12712, 13121, 13488, 16804, 17584, 28709
863	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 838, 850, 852, 857, 860-861, 865, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1041-1043, 1051, 1097, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 28709
864	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 720, 819, 833, 838, 850, 870-871, 875, 880, 905, 918, 1008, 1025-1027, 1046, 1089, 1097, 1127, 1200, 1208, 1252, 1256, 4386, 4929, 4934, 4946, 4960, 4971, 5104, 5123, 5142, 5352, 8229, 8482, 8612, 9025, 9030, 9056, 9238, 13121, 13488, 16804, 17248, 17584, 28709
865	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 819, 833, 838, 850, 858, 860, 863, 870-871, 875, 880, 905, 923-924, 1025-1027, 1097, 1142-1143, 1148, 1200, 1208, 1252, 4386, 4929, 4934, 4946, 4971, 5123, 5348, 8229, 8482, 8612, 9025, 9030, 13121, 13488, 16804, 17584, 28709
866	37, 256, 437, 500, 819, 850, 855, 870, 878, 880, 915, 1025, 1200, 1208, 1251-1252, 1283, 4946, 4951, 5347, 8229, 13488, 17584, 28709
867	259, 1153-1155, 1160, 1200, 1208, 4899, 5351, 9048, 12712, 13488, 17584

z/OS conversion support

Table 102. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
868	918, 1006, 1200, 1208, 13488, 17584
869	37, 256, 259, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 813, 819, 838, 850, 852, 857, 860-861, 863, 870-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252-1254, 1280, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5349, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
870	37, 256, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-866, 869, 871, 874-875, 880, 897, 903, 912, 915-916, 920, 1009, 1025-1027, 1040-1043, 1047, 1088, 1112, 1122, 1200, 1208, 1250, 1252, 1282, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5346, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
871	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869, 870, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
872	259, 808, 858-859, 923-924, 1140-1149, 1153-1155, 1200, 1208, 5347, 5348, 13488, 17584
874	37, 259, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
875	37, 256, 273, 277-278, 280, 284-285, 297, 367, 423, 437, 500, 737, 775, 813, 819, 836, 838, 850-852, 857, 860-865, 869-871, 874, 880, 897, 903, 912, 916, 920, 1009, 1025-1027, 1041-1043, 1047, 1088, 1112, 1122, 1200, 1208, 1252-1253, 1280, 4909, 4932, 4934, 4946, 4948, 4953, 4960, 4970-4971, 5012, 5123, 5349, 8229, 9030, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
878	855, 866, 880, 915, 1025, 1131, 1200, 1208, 1251, 1283, 4951, 5347, 13488, 17584
880	37, 256, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 775, 813, 819, 838, 850, 852, 855, 857, 860-866, 869-871, 874-875, 878, 897, 903, 912, 915-916, 920, 1009, 1025-1027, 1041-1043, 1112, 1122, 1200, 1208, 1251-1252, 1283, 4909, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5347, 8229, 9030, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 28709
891	500, 833, 1088, 1200, 1208, 4929, 9025, 13121, 13488, 17584, 25664
895	290, 500, 1027, 1041, 1200, 1208, 4386, 5123, 8482, 13488, 17584, 25617
896	290, 1027, 1041, 1200, 1208, 4386, 4992, 5123, 8482, 13488, 17584, 25617
897	37, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252, 4386, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 8229, 8482, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
899	259
901	259, 858-859, 902, 923-924, 1140, 1148, 1156-1157, 1200, 1208, 5348, 5353, 13488, 17584
902	259, 858-859, 901, 923-924, 1140, 1148, 1156-1157, 1200, 1208, 5348, 5353, 13488, 17584
903	37, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 836, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 912, 916, 920, 1025-1027, 1041-1043, 1115, 1200, 1208, 1252, 4909, 4932, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5211, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
904	37, 500, 1114, 1200, 1208, 5210, 8229, 13488, 17584, 25480, 28709

Table 102. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
905	37, 256, 437, 500, 737, 775, 819, 850, 852, 857, 860-865, 920, 1026, 1112, 1122, 1200, 1208, 1252, 1254, 1281, 4946, 4948, 4953, 4960, 8229, 9044, 9049, 9056, 13488, 17248, 17584, 28709
912	37, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 916, 920, 1025-1027, 1041-1043, 1047, 1200, 1208, 1250, 1252, 1282, 4909, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5346, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
914	37, 437, 500, 819, 850, 1200, 1208, 1252, 1257, 4946, 8229, 13488, 17584, 28709
915	37, 259, 437, 500, 819, 850, 855, 866, 870, 878, 880, 1025, 1131, 1200, 1208, 1251-1252, 1283, 4946, 4951, 5347, 8229, 13488, 17584, 28709
916	37, 273, 277-278, 280, 284-285, 297, 423-424, 437, 500, 803, 813, 819, 838, 850, 852, 856-857, 860-863, 869-871, 874-875, 880, 897, 903, 912, 920, 1025-1027, 1041-1043, 1200, 1208, 1252, 1255, 4909, 4934, 4946, 4948, 4952-4953, 4970-4971, 5012, 5123, 5351, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
918	864, 868, 1006, 1200, 1208, 4960, 9056, 13488, 17248, 17584
920	37, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 1025-1026, 1200, 1208, 1252, 1254, 1281, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5350, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 28709
921	37, 437, 500, 819, 850, 922, 1112, 1122, 1200, 1208, 1252, 1257, 4946, 5353, 8229, 13488, 17584, 28709
922	37, 437, 500, 819, 850, 921, 1112, 1122, 1200, 1208, 1252, 1257, 4946, 5353, 8229, 13488, 17584, 28709
923	37, 273, 277-278, 280, 284-285, 297, 437, 500, 808, 819, 850, 858, 860-861, 865, 871-872, 901-902, 924, 1047, 1051, 1140-1149, 1153-1158, 1160-1162, 1164, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
924	37, 273, 277-278, 280, 284-285, 297, 437, 500, 808, 819, 848-850, 858, 860-861, 865, 871-872, 901-902, 923, 1047, 1051, 1140-1149, 1153-1157, 1160-1164, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
926	834, 951, 9026
927	835, 947, 1200, 1208, 4931, 9027, 13488, 17584, 21427
928	837, 1200, 1208, 1380, 13488, 17584
930	931-932, 939, 942-943, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
931	930, 932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
932	930-931, 939, 942-943, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
933	934, 944, 949, 1200, 1208, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13488, 13651, 17317, 17584, 25510, 25520, 25525, 29616, 29621, 33717, 37813
934	933, 949, 5029, 5045, 5460, 9125, 13221, 17317, 25510, 25525, 29621, 33717, 37813
935	936, 946, 1200, 1208, 1381, 1386, 1388, 5031, 5477, 5482, 5484, 9127, 13223, 13488, 17584, 25512
936	935, 946, 1381, 5031, 5477, 5484, 9127, 13223, 25512
937	938, 948, 950, 1200, 1208, 1370, 5033, 5046, 9142, 13488, 17584, 25514, 25524, 29620

z/OS conversion support

Table 102. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
938	937, 950, 1370, 5033, 5046, 9142, 25514
939	930-932, 942-943, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
941	300-301, 1200, 1208, 1351, 4396, 8492, 13488, 16684, 17584
942	930-932, 939, 943, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
943	930-932, 939, 942, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
944	933, 949, 1200, 1208, 5029, 5045, 5460, 9125, 13221, 13488, 17317, 17584, 25520, 25525, 29616, 29621, 33717, 37813
946	935-936, 1200, 1208, 5031, 5484, 9127, 13223, 13488, 17584, 25512
947	835, 927, 1200, 1208, 4931, 9027, 13488, 17584, 21427
948	937, 950, 1200, 1208, 1370, 5033, 5046, 9142, 13488, 17584, 25524, 29620
949	933-934, 944, 1200, 1208, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13488, 13651, 17317, 17584, 25510, 25520, 25525, 29616, 29621, 33717, 37813
950	937-938, 948, 1200, 1208, 1370, 5033, 5046, 9142, 13488, 17584, 25514, 25524, 29620
951	834, 926, 1200, 1208, 1362, 4930, 9026, 13488, 17584
1004	500, 819, 850, 1200, 1208, 4946, 13488, 17584
1006	868, 918, 1200, 1208, 13488, 17584
1008	420, 864, 1200, 1208, 4960, 5104, 8612, 9056, 13488, 16804, 17248, 17584
1009	37, 273, 277-278, 280, 284, 290, 297, 367, 423, 500, 833, 836, 870-871, 875, 880, 1025-1026, 1200, 1208, 4386, 4929, 4932, 4971, 8229, 8482, 9025, 13121, 13488, 17584, 28709
1010	500, 1200, 1208, 13488, 17584
1011	500, 1200, 1208, 13488, 17584
1012	500, 1200, 1208, 13488, 17584
1013	500, 1140, 1200, 1208, 13488, 17584
1014	500, 1200, 1208, 13488, 17584
1015	500, 1200, 1208, 13488, 17584
1016	500, 1200, 1208, 13488, 17584
1017	500, 1200, 1208, 13488, 17584
1018	500, 1200, 1208, 13488, 17584
1019	500, 1200, 1208, 13488, 17584
1020	500
1021	500
1023	500
1025	37, 256, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-866, 869-871, 874-875, 878, 880, 897, 903, 912, 915-916, 920, 1009, 1026-1027, 1040-1043, 1051, 1088, 1112, 1122, 1131, 1200, 1208, 1251-1252, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5347, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709

Table 102. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
1026	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-865, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1009, 1025, 1027, 1040-1043, 1047, 1088, 1112, 1122, 1200, 1208, 1252, 1254, 1281, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5350, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
1027	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-865, 869-871, 874-875, 880, 895-897, 903, 912, 916, 1025-1026, 1040-1043, 1047, 1088, 1112, 1122, 1139, 1200, 1208, 1252, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 4992, 5012, 5123, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
1040	37, 273, 277-278, 280, 284-285, 290, 297, 437, 500, 833, 836, 850, 852, 855, 857, 870-871, 1025-1027, 1041-1043, 1088, 1200, 1208, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 5123, 8229, 8482, 9025, 9044, 9049, 13121, 13488, 17584, 25617, 25619, 25664, 28709
1041	37, 273, 277-278, 280, 284-285, 290, 297, 367, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 895-897, 903, 912, 916, 1025-1027, 1040, 1042-1043, 1088, 1200, 1208, 1252, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 4992, 5012, 5123, 8229, 8482, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 17584, 25473, 25479, 25617, 25619, 25664, 28709
1042	37, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 1025-1027, 1040, 1041, 1043, 1088, 1200, 1208, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 8229, 8482, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 17584, 25473, 25479, 25617, 25619, 25664, 28709
1043	37, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 1025-1027, 1040, 1041, 1042, 1088, 1114, 1200, 1208, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5210, 8229, 8482, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 17584, 25473, 25479, 25617, 25619, 25664, 28709
1046	420, 500, 864, 1089, 1127, 1200, 1208, 1256, 4960, 5142, 5352, 8612, 9056, 9238, 13488, 16804, 17248, 17584
1047	37, 273-275, 277-278, 280, 281, 282, 284-285, 297, 437, 500, 819, 850, 852, 858, 870-871, 875, 912, 923-924, 1026-1027, 1140-1149, 1200, 1208, 1252, 1254, 4946, 4948, 4971, 5123, 8229, 9044, 13488, 17584, 28709
1051	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 871, 923-924, 1025, 1097, 1140-1149, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1088	37, 273, 277-278, 280, 284-285, 290, 297, 367, 500, 819, 833, 836, 850, 852, 855, 857, 870-871, 875, 891, 1025-1027, 1040-1043, 1126, 1200, 1208, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4971, 5123, 8229, 8482, 9025, 9044, 9049, 13121, 13488, 17584, 25617, 25619, 25664, 28709
1089	420, 500, 819, 850, 864, 1046, 1127, 1200, 1208, 1256, 4946, 4960, 5142, 5352, 8612, 9056, 9238, 13488, 16804, 17248, 17584
1097	37, 437, 500, 737, 775, 819, 850, 852, 857, 860-865, 1051, 1098, 1112, 1122, 1200, 1208, 1252, 4946, 4948, 4953, 4960, 8229, 9044, 9049, 9056, 13488, 17248, 17584, 28709
1098	259, 420, 437, 819, 850, 1097, 1200, 1208, 1252, 4946, 8612, 13488, 16804, 17584
1100	37, 273, 277-278, 280, 284-285, 297, 500, 850, 4946, 8229, 28709
1101	500
1102	500
1103	500

z/OS conversion support

Table 102. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
1104	500
1105	500
1106	500
1107	500
1112	37, 256, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 775, 819, 833, 836, 838, 850, 870-871, 875, 880, 905, 921-922, 1025-1027, 1097, 1122, 1200, 1208, 1252, 1257, 4386, 4929, 4932, 4934, 4946, 4971, 5123, 5353, 8229, 8482, 8612, 9025, 9030, 13121, 13488, 16804, 17584, 28709
1114	37, 437, 500, 819, 836, 850, 904, 1043, 1115, 1200, 1208, 4932, 4946, 5210-5211, 8229, 13488, 17584, 25480, 25619, 28709
1115	37, 367, 437, 500, 836, 903, 1114, 1200, 1208, 4932, 5210-5211, 8229, 13488, 17584, 25479, 28709
1122	37, 256, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 775, 819, 833, 836, 838, 850, 870-871, 875, 880, 905, 921-922, 1025-1027, 1097, 1112, 1200, 1208, 1252, 1257, 4386, 4929, 4932, 4934, 4946, 4971, 5123, 5353, 8229, 8482, 8612, 9025, 9030, 13121, 13488, 16804, 17584, 28709
1123	819, 1124-1125, 1148, 1200, 1208, 1251-1252, 1283, 5347, 13488, 17584
1124	37, 500, 1123, 1125, 1200, 1208, 1251, 1283, 5347, 8229, 13488, 17584, 28709
1125	500, 1123, 1124, 1200, 1208, 1251, 1283, 5347, 13488, 17584
1126	37, 367, 437, 500, 819, 833, 850, 1088, 1200, 1208, 1252, 4929, 4946, 8229, 9025, 13121, 13488, 17584, 25664, 28709
1127	420, 864, 1046, 1089, 1256, 4960, 5142, 8612, 9056, 9238, 16804, 17248
1129	500, 1130, 1200, 1208, 1258, 5354, 13488, 17584
1130	37, 500, 819, 850, 1129, 1200, 1208, 1252, 1258, 4946, 5354, 8229, 13488, 17584, 28709
1131	37, 500, 878, 915, 1025, 1200, 1208, 1251, 1283, 5347, 8229, 13488, 17584, 28709
1132	37, 500, 819, 850, 1133, 1200, 1208, 1252, 4946, 8229, 13488, 17584, 28709
1133	500, 1132, 1200, 1208, 13488, 17584
1137	37, 500, 819, 1200, 1208, 8229, 13488, 17584, 28709
1139	290, 1027, 4386, 5123, 8482
1140	37, 273, 277-278, 280, 284-285, 297, 437, 500, 808, 819, 850, 858, 860, 863, 871-872, 901-902, 923-924, 1013, 1047, 1051, 1141-1149, 1153-1157, 1160-1162, 1164, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1141	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 871-872, 923-924, 1047, 1051, 1140, 1142-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1142	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 865, 871-872, 923-924, 1047, 1051, 1140-1141, 1143-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1143	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 865, 871-872, 923-924, 1047, 1051, 1140-1142, 1144-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1144	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 871-872, 923-924, 1047, 1051, 1140-1143, 1145-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709

Table 102. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
1145	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 860, 863, 871-872, 923-924, 1047, 1051, 1140-1144, 1146-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1146	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 860, 863, 871-872, 923-924, 1047, 1051, 1140-1145, 1147-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1147	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 871-872, 923-924, 1047, 1051, 1140-1146, 1148-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1148	37, 273, 277-278, 280, 284-285, 297, 437, 500, 808, 819, 848-850, 858, 860-861, 863, 865, 871-872, 901-902, 923-924, 1047, 1051, 1123, 1140-1147, 1149, 1153-1164, 1200, 1208, 1252, 1275, 4899, 4946, 5348, 5349, 8229, 12712, 13488, 17584, 28709
1149	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 861, 863, 871-872, 923-924, 1047, 1051, 1140-1148, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1153	808, 858-859, 867, 872, 923-924, 1140-1149, 1154-1157, 1160-1162, 1200, 1208, 5348, 9044, 13488, 17584
1154	808, 849, 858-859, 867, 872, 923-924, 1140-1149, 1153, 1155-1157, 1160-1162, 1200, 1208, 5347, 5348, 13488, 17584
1155	858-859, 867, 872, 923-924, 1140-1149, 1153-1154, 1156-1157, 1160-1162, 1200, 1208, 5348, 5350, 9049, 13488, 17584
1156	858-859, 901-902, 923-924, 1140-1149, 1153-1155, 1157, 1160, 1200, 1208, 5348, 5353, 12712, 13488, 17584
1157	858-859, 901-902, 923-924, 1140-1149, 1153-1156, 1160, 1200, 1208, 5348, 5353, 12712, 13488, 17584
1158	848, 923, 1148, 1200, 1208, 5347, 5348, 13488, 17584
1159	1148, 1200, 1208, 13488, 17584
1160	858-859, 867, 923-924, 1140-1149, 1153-1157, 1161-1162, 1200, 1208, 5348, 13488, 17584
1161	259, 858-859, 923-924, 1140-1149, 1153-1155, 1160, 5348, 17584
1162	259, 858-859, 923-924, 1140-1149, 1153-1155, 1160, 5348, 17584
1163	924, 1148, 1164, 5354, 17584
1164	858-859, 923-924, 1140, 1148, 1163, 1200, 1208, 5348, 5354, 13488, 17584
1200	37, 256, 259, 273, 275, 277-278, 280, 282, 284-285, 290, 293, 297, 300-301, 367, 420, 423-424, 437, 500, 720, 737, 775, 803, 806, 808, 813, 819, 833-838, 848-852, 855-872, 874-875, 878, 880, 891, 895-897, 901-905, 912, 914-916, 918, 920-924, 927-928, 930, 932-933, 935, 937, 939, 941-944, 946-951, 1004, 1006, 1008-1019, 1025-1027, 1040-1043, 1046-1047, 1051, 1088-1089, 1097-1098, 1112, 1114-1115, 1122-1126, 1129-1133, 1137, 1140-1149, 1153-1160, 1164, 1208, 1250-1258, 1275-1277, 1280-1285, 1351, 1362-1364, 1370-1371, 1380-1381, 1385-1386, 1388, 1390, 1399, 4899, 4909, 4930, 4933, 4948, 4951-4952, 4960, 4971, 5012, 5039, 5104, 5123, 5142, 5210, 5346-5354, 8482, 8612, 9027, 9030, 9044, 9048-9049, 9056, 9061, 9066, 9238, 12712, 13121, 13218, 13488, 16684, 16804, 17248, 17584, 21427, 28709

z/OS conversion support

Table 102. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
1208	37, 256, 259, 273, 275, 277-278, 280, 282, 284-285, 290, 293, 297, 300-301, 367, 420, 423-424, 437, 500, 720, 737, 775, 803, 806, 808, 813, 819, 833-838, 848-852, 855-872, 874-875, 878, 880, 891, 895-897, 901-905, 912, 914-916, 918, 920-924, 927-928, 930, 932-933, 935, 937, 939, 941-944, 946-951, 1004, 1006, 1008-1019, 1025-1027, 1040-1043, 1046-1047, 1051, 1088-1089, 1097-1098, 1112, 1114-1115, 1122-1126, 1129-1133, 1137, 1140-1149, 1153-1160, 1164, 1200, 1250-1258, 1275-1277, 1280-1285, 1351, 1362-1364, 1370-1371, 1380-1381, 1385-1386, 1388, 1390, 1399, 4899, 4909, 4930, 4933, 4948, 4951-4952, 4960, 4971, 5012, 5039, 5104, 5123, 5142, 5210, 5346-5354, 8482, 8612, 9027, 9030, 9044, 9048-9049, 9056, 9061, 9066, 9238, 12712, 13121, 13218, 13488, 16684, 16804, 17248, 17584, 21427, 28709
1250	37, 259, 273, 500, 819, 850, 852, 855, 870, 912, 1200, 1208, 1252, 1282, 4946, 4948, 4951, 5346, 8229, 9044, 13488, 17584, 28709
1251	37, 256, 259, 500, 819, 850, 855, 866, 878, 880, 915, 1025, 1123-1125, 1131, 1200, 1208, 1252, 1283, 4946, 4951, 5347, 8229, 13488, 17584, 28709
1252	37, 256, 259, 273, 275, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 737, 775, 803, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-866, 869-871, 874-875, 880, 897, 903, 905, 912, 914-916, 920-924, 1025-1027, 1041, 1047, 1051, 1097-1098, 1112, 1122-1123, 1126, 1130, 1132, 1140-1149, 1200, 1208, 1250-1251, 1254-1255, 1257, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 16804, 17248, 17584, 25473, 25479, 25617, 28709
1253	37, 259, 423, 500, 737, 813, 819, 850, 869, 875, 1200, 1208, 1280, 4909, 4946, 4971, 5349, 8229, 9061, 13488, 17584, 28709
1254	37, 259, 500, 819, 850, 857, 869, 905, 920, 1026, 1047, 1200, 1208, 1252, 1281, 4946, 4953, 5350, 8229, 9049, 9061, 13488, 17584, 28709
1255	37, 259, 424, 500, 803, 819, 850, 856, 862, 916, 1200, 1208, 1252, 1281, 4946, 4952, 5012, 5351, 8229, 13488, 17584, 28709
1256	259, 420, 500, 720, 850, 864, 1046, 1089, 1127, 1200, 1208, 4946, 4960, 5142, 5352, 8612, 9056, 9238, 13488, 16804, 17248, 17584
1257	37, 259, 437, 500, 775, 819, 850, 914, 921-922, 1112, 1122, 1200, 1208, 1252, 4946, 5353, 8229, 13488, 17584, 28709
1258	37, 259, 500, 819, 1129-1130, 1200, 1208, 5354, 8229, 13488, 17584, 28709
1275	37, 256, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 871, 923-924, 1051, 1140-1149, 1200, 1208, 1252, 4946, 5348, 8229, 13488, 17584, 28709
1276	1200, 1208, 13488, 17584
1277	1200, 1208, 13488, 17584
1280	37, 423, 437, 500, 737, 813, 819, 850, 869, 875, 1200, 1208, 1252-1253, 4909, 4946, 4971, 5349, 8229, 9061, 13488, 17584, 28709
1281	37, 437, 500, 819, 850, 857, 905, 920, 1026, 1200, 1208, 1252, 1254-1255, 4946, 4953, 5350, 8229, 9049, 13488, 17584, 28709
1282	500, 852, 870, 912, 1200, 1208, 1250, 4948, 5346, 9044, 13488, 17584
1283	37, 437, 500, 819, 850, 855, 866, 878, 880, 915, 1025, 1123-1125, 1131, 1200, 1208, 1251-1252, 4946, 4951, 5347, 8229, 13488, 17584, 28709
1284	1200, 1208, 13488, 17584
1285	1200, 1208, 13488, 17584
1351	300-301, 941, 1200, 1208, 4396, 8492, 13488, 16684, 17584
1362	834, 951, 1200, 1208, 4930, 9026, 13488, 17584

Table 102. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
1363	933, 949, 1200, 1208, 1364, 5029, 5045, 5460, 9125, 9555, 13221, 13488, 13651, 17317, 17584, 25525, 29621, 33717, 37813
1364	933, 949, 1200, 1208, 1363, 5029, 5045, 5460, 9125, 9555, 13221, 13488, 13651, 17317, 17584, 25525, 29621, 33717, 37813
1370	937-938, 948, 950, 1200, 1208, 1371, 5033, 5046, 9142, 13488, 17584, 25514, 25524, 29620
1371	1200, 1208, 1370, 13488, 17584
1380	837, 928, 1200, 1208, 1385, 4933, 13488, 17584
1381	935-936, 1200, 1208, 1386, 1388, 5031, 5477, 5482, 5484, 9127, 13223, 13488, 17584, 25512
1385	837, 1200, 1208, 1380, 4933, 13488, 17584
1386	935, 1200, 1208, 1381, 1388, 5031, 5477, 5482, 5484, 9127, 13223, 13488, 17584
1388	935, 1200, 1208, 1381, 1386, 5031, 5477, 5482, 5484, 5488, 9127, 13223, 13488, 17584
1390	930-932, 939, 942-943, 1200, 1208, 1399, 5026, 5028, 5035, 5038-5039, 5055, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
1399	930-932, 939, 942-943, 1200, 1208, 1390, 5026, 5028, 5035, 5038-5039, 5050, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
4386	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 860-865, 870-871, 895-897, 1009, 1025-1027, 1040-1043, 1088, 1112, 1122, 1139, 1252, 4929, 4932, 4946, 4948, 4951, 4953, 4960, 4992, 5123, 8229, 8482, 9025, 9044, 9049, 9056, 13121, 17248, 25473, 25617, 25619, 25664, 28709
4396	300-301, 941, 1351, 8492, 16684
4899	867, 1148, 1200, 1208, 5351, 9048, 12712, 13488, 17584
4909	37, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252-1253, 1280, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5349, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
4929	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 860-865, 870-871, 891, 1009, 1025-1027, 1040-1043, 1088, 1112, 1122, 1126, 1252, 4386, 4932, 4946, 4948, 4951, 4953, 4960, 5123, 8229, 8482, 9025, 9044, 9049, 9056, 13121, 17248, 25617, 25619, 25664, 28709
4930	834, 951, 1200, 1208, 1362, 9026, 13488, 17584
4931	835, 927, 947, 9027, 21427
4932	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 424, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 870-871, 875, 903, 1009, 1025-1027, 1040-1043, 1088, 1112, 1114-1115, 1122, 1252, 4386, 4929, 4946, 4948, 4951, 4953, 4971, 5123, 5210-5211, 8229, 8482, 9025, 9044, 9049, 13121, 25479, 25617, 25619, 25664, 28709
4933	837, 1200, 1208, 1380, 1385, 13488, 17584
4934	37, 256, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 775, 813, 819, 838, 850, 852, 857, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1112, 1122, 1252, 4909, 4946, 4948, 4953, 4960, 4970-4971, 5012, 5123, 8229, 9030, 9044, 9049, 9056, 9061, 9066, 17248, 25473, 25479, 25617, 25619, 28709

z/OS conversion support

Table 102. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
4946	37, 256, 259, 273, 275, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 737, 775, 803, 813, 819, 833, 836, 838, 850, 852, 855-858, 860-866, 869-871, 874-875, 880, 897, 903, 905, 912, 914-916, 920-924, 1004, 1025-1027, 1040-1043, 1047, 1051, 1088-1089, 1097-1098, 1100, 1112, 1114, 1122, 1126, 1130, 1132, 1140-1149, 1250-1257, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4948, 4951-4953, 4960, 4970-4971, 5012, 5123, 5210, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 16804, 17248, 25473, 25479, 25617, 25619, 25664, 28709
4948	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1040-1043, 1047, 1088, 1097, 1200, 1208, 1250, 1252, 1282, 4386, 4909, 4929, 4932, 4934, 4946, 4951, 4953, 4970-4971, 5012, 5123, 5346, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 25664, 28709
4951	37, 259, 273, 277-278, 280, 284-285, 290, 297, 437, 500, 819, 833, 836, 850, 852, 855, 857, 866, 870-871, 878, 880, 912, 915, 1025-1027, 1040-1043, 1088, 1200, 1208, 1250-1252, 1283, 4386, 4929, 4932, 4946, 4948, 4953, 5123, 5346, 5347, 8229, 8482, 9025, 9044, 9049, 13121, 13488, 17584, 25617, 25619, 25664, 28709
4952	259, 273, 424, 500, 803, 850, 856, 862, 916, 1200, 1208, 1255, 4946, 5012, 5351, 13488, 17584
4953	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1040-1043, 1088, 1097, 1252, 1254, 1281, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4970-4971, 5012, 5123, 5350, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 16804, 25473, 25479, 25617, 25619, 25664, 28709
4960	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 720, 819, 833, 838, 850, 864, 870-871, 875, 880, 905, 918, 1008, 1025-1027, 1046, 1089, 1097, 1127, 1200, 1208, 1252, 1256, 4386, 4929, 4934, 4946, 4971, 5104, 5123, 5142, 5352, 8229, 8482, 8612, 9025, 9030, 9056, 9238, 13121, 13488, 16804, 17248, 17584, 28709
4970	37, 259, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1252, 4909, 4934, 4946, 4948, 4953, 4971, 5012, 5123, 8229, 9030, 9044, 9049, 9061, 9066, 25473, 25479, 25617, 25619, 28709
4971	37, 256, 273, 277-278, 280, 284-285, 297, 367, 423, 437, 500, 737, 775, 813, 819, 836, 838, 850-852, 857, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1009, 1025-1027, 1041-1043, 1047, 1088, 1112, 1122, 1200, 1208, 1252-1253, 1280, 4909, 4932, 4934, 4946, 4948, 4953, 4960, 4970, 5012, 5123, 5349, 8229, 9030, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
4992	290, 896, 1027, 1041, 4386, 5123, 8482, 25617
5012	37, 273, 277-278, 280, 284-285, 297, 423-424, 437, 500, 803, 813, 819, 838, 850, 852, 856-857, 860-863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252, 1255, 4909, 4934, 4946, 4948, 4952-4953, 4970-4971, 5123, 5351, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
5026	930-932, 939, 942-943, 1390, 1399, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
5028	930-932, 939, 942-943, 1390, 1399, 5026, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
5029	933-934, 944, 949, 1363-1364, 5045, 5460, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 33717, 37813
5031	935-936, 946, 1381, 1386, 1388, 5477, 5482, 5484, 9127, 13223, 25512
5033	937-938, 948, 950, 1370, 5046, 9142, 25514, 25524, 29620

Table 102. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
5035	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
5038	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
5039	930-932, 939, 942-943, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
5045	933-934, 944, 949, 1363-1364, 5029, 5460, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 33717, 37813
5046	937-938, 948, 950, 1370, 5033, 9142, 25514, 25524, 29620
5104	420, 864, 1008, 1200, 1208, 4960, 8612, 9056, 13488, 16804, 17248, 17584
5123	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-865, 869-871, 874-875, 880, 895-897, 903, 912, 916, 1025-1027, 1040-1043, 1047, 1088, 1112, 1122, 1139, 1200, 1208, 1252, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 4992, 5012, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
5142	420, 500, 864, 1046, 1089, 1127, 1200, 1208, 1256, 4960, 5352, 8612, 9056, 9238, 13488, 16804, 17248, 17584
5210	37, 437, 500, 819, 836, 850, 904, 1043, 1114-1115, 1200, 1208, 4932, 4946, 5211, 8229, 13488, 17584, 25480, 25619, 28709
5211	37, 367, 437, 500, 836, 903, 1114-1115, 4932, 5210, 8229, 25479, 28709
5346	37, 259, 273, 500, 819, 850, 852, 855, 870, 912, 1200, 1208, 1250, 1252, 1282, 4946, 4948, 4951, 8229, 9044, 13488, 17584, 28709
5347	808, 848-849, 855, 866, 872, 878, 880, 915, 1025, 1123-1125, 1131, 1154, 1158, 1200, 1208, 1251, 1283, 4951, 13488, 17584
5348	37, 259, 273, 275, 277-278, 280, 284-285, 297, 437, 500, 808, 819, 850, 858, 860-861, 863, 865, 871-872, 901-902, 923-924, 1051, 1140-1149, 1153-1158, 1160-1162, 1164, 1200, 1208, 1252, 1275, 4946, 8229, 13488, 17584, 28709
5349	813, 869, 875, 1148, 1200, 1208, 1253, 1280, 4909, 4971, 9061, 13488, 17584
5350	857, 920, 1026, 1155, 1200, 1208, 1254, 1281, 4953, 9049, 13488, 17584
5351	424, 856, 862, 867, 916, 1200, 1208, 1255, 4899, 4952, 5012, 9048, 12712, 13488, 17584
5352	420, 864, 1046, 1089, 1200, 1208, 1256, 4960, 5142, 8612, 9056, 9238, 13488, 16804, 17248, 17584
5353	901-902, 921-922, 1112, 1122, 1156-1157, 1200, 1208, 1257, 13488, 17584
5354	1129-1130, 1163, 1164, 1200, 1208, 1258, 13488, 17584
5460	933-934, 944, 949, 1363-1364, 5029, 5045, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 33717, 37813
5477	935-936, 1381, 1386, 1388, 5031, 5482, 5484, 9127, 13223, 25512
5482	935, 1381, 1386, 1388, 5031, 5477, 5484, 9127, 13223
5484	935-936, 946, 1381, 1386, 1388, 5031, 5477, 5482, 9127, 13223, 25512
5488	1388

z/OS conversion support

Table 102. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
8229	37, 256, 273, 275, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 720, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-866, 869-871, 874-875, 880, 897, 903-905, 912, 914-916, 920-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1097, 1100, 1112, 1114-1115, 1122, 1124, 1126, 1130-1132, 1137, 1140-1149, 1250-1255, 1257-1258, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5210-5211, 5346, 5348, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 16804, 17248, 25473, 25479, 25480, 25617, 25619, 25664, 28709
8482	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 860-865, 870-871, 895-897, 1009, 1025-1027, 1040-1043, 1088, 1112, 1122, 1139, 1200, 1208, 1252, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4960, 4992, 5123, 8229, 9025, 9044, 9049, 9056, 13121, 13488, 17248, 17584, 25473, 25617, 25619, 25664, 28709
8492	300-301, 941, 1351, 4396, 16684
8612	37, 256, 420, 424, 437, 500, 720, 737, 775, 819, 850, 852, 857, 860-865, 1008, 1046, 1089, 1098, 1112, 1122, 1127, 1200, 1208, 1252, 1256, 4946, 4948, 4953, 4960, 5104, 5142, 5352, 8229, 9044, 9049, 9056, 9238, 13488, 16804, 17248, 17584, 28709
9025	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 860-865, 870-871, 891, 1009, 1025-1027, 1040-1043, 1088, 1112, 1122, 1126, 1252, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4960, 5123, 8229, 8482, 9044, 9049, 9056, 13121, 17248, 25617, 25619, 25664, 28709
9026	834, 926, 951, 1362, 4930
9027	835, 927, 947, 1200, 1208, 4931, 13488, 17584, 21427
9030	37, 256, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 775, 813, 819, 838, 850, 852, 857, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1112, 1122, 1200, 1208, 1252, 4909, 4934, 4946, 4948, 4953, 4960, 4970-4971, 5012, 5123, 8229, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 28709
9044	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1040-1043, 1047, 1088, 1097, 1153, 1200, 1208, 1250, 1252, 1282, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5346, 8229, 8482, 8612, 9025, 9030, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 25664, 28709
9048	867, 1200, 1208, 4899, 5351, 12712, 13488, 17584
9049	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1040-1043, 1088, 1097, 1155, 1200, 1208, 1252, 1254, 1281, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5350, 8229, 8482, 8612, 9025, 9030, 9044, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 25664, 28709
9056	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 720, 819, 833, 838, 850, 864, 870-871, 875, 880, 905, 918, 1008, 1025-1027, 1046, 1089, 1097, 1127, 1200, 1208, 1252, 1256, 4386, 4929, 4934, 4946, 4960, 4971, 5104, 5123, 5142, 5352, 8229, 8482, 8612, 9025, 9030, 9238, 13121, 13488, 16804, 17248, 17584, 28709
9061	37, 256, 259, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252-1254, 1280, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5349, 8229, 9030, 9044, 9049, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
9066	37, 259, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 8229, 9030, 9044, 9049, 9061, 13488, 17584, 25473, 25479, 25617, 25619, 28709

Table 102. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
9122	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
9124	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
9125	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 33717, 37813
9127	935-936, 946, 1381, 1386, 1388, 5031, 5477, 5482, 5484, 13223, 25512
9131	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
9135	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
9142	937-938, 948, 950, 1370, 5033, 5046, 25514, 25524, 29620
9238	420, 500, 864, 1046, 1089, 1127, 1200, 1208, 1256, 4960, 5142, 5352, 8612, 9056, 13488, 16804, 17248, 17584
9555	933, 949, 1363-1364, 5029, 5045, 5460, 9125, 13221, 13651, 17317, 25525, 29621, 33717, 37813
12712	862, 867, 1148, 1156-1157, 1200, 1208, 4899, 5351, 9048, 13488, 17584
13121	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 860-865, 870-871, 891, 1009, 1025-1027, 1040-1043, 1088, 1112, 1122, 1126, 1200, 1208, 1252, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4960, 5123, 8229, 8482, 9025, 9044, 9049, 9056, 13488, 17248, 17584, 25617, 25619, 25664, 28709
13218	930-932, 939, 942-943, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
13219	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
13221	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 33717, 37813
13223	935-936, 946, 1381, 1386, 1388, 5031, 5477, 5482, 5484, 9127, 25512
13231	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 17314, 25508, 25518, 29614, 33698-33700, 37796
13488	37, 256, 259, 273, 275, 277-278, 280, 282, 284-285, 290, 293, 297, 300-301, 367, 420, 423-424, 437, 500, 720, 737, 775, 803, 806, 808, 813, 819, 833-838, 848-852, 855-872, 874-875, 878, 880, 891, 895-897, 901-905, 912, 914-916, 918, 920-924, 927-928, 930, 932-933, 935, 937, 939, 941-944, 946-951, 1004, 1006, 1008-1019, 1025-1027, 1040-1043, 1046-1047, 1051, 1088-1089, 1097-1098, 1112, 1114-1115, 1122-1126, 1129-1133, 1137, 1140-1149, 1153-1160, 1164, 1200, 1208, 1250-1258, 1275-1277, 1280-1285, 1351, 1362-1364, 1370-1371, 1380-1381, 1385-1386, 1388, 1390, 1399, 4899, 4909, 4930, 4933, 4948, 4951-4952, 4960, 4971, 5012, 5039, 5104, 5123, 5142, 5210, 5346-5354, 8482, 8612, 9027, 9030, 9044, 9048-9049, 9056, 9061, 9066, 9238, 12712, 13121, 13218, 16684, 16804, 17248, 17584, 21427, 28709
13651	933, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 17317, 25525, 29621, 33717, 37813
16684	300-301, 941, 1200, 1208, 1351, 4396, 8492, 13488, 17584
16804	37, 256, 420, 424, 437, 500, 720, 737, 775, 819, 850, 852, 857, 860-865, 1008, 1046, 1089, 1098, 1112, 1122, 1127, 1200, 1208, 1252, 1256, 4946, 4948, 4953, 4960, 5104, 5142, 5352, 8229, 8612, 9044, 9049, 9056, 9238, 13488, 17248, 17584, 28709

z/OS conversion support

Table 102. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
17248	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 720, 819, 833, 838, 850, 864, 870-871, 875, 880, 905, 918, 1008, 1025-1027, 1046, 1089, 1097, 1127, 1200, 1208, 1252, 1256, 4386, 4929, 4934, 4946, 4960, 4971, 5104, 5123, 5142, 5352, 8229, 8482, 8612, 9025, 9030, 9056, 9238, 13121, 13488, 16804, 17584, 28709
17314	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 25508, 25518, 29614, 33698-33700, 37796
17317	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13651, 25510, 25520, 25525, 29616, 29621, 33717, 37813
17584	37, 256, 259, 273, 275, 277-278, 280, 282, 284-285, 290, 293, 297, 300-301, 367, 420, 423-424, 437, 500, 720, 737, 775, 803, 806, 808, 813, 819, 833-838, 848-852, 855-872, 874-875, 878, 880, 891, 895-897, 901-905, 912, 914-916, 918, 920-924, 927-928, 930, 932-933, 935, 937, 939, 941-944, 946-951, 1004, 1006, 1008-1019, 1025-1027, 1040-1043, 1046-1047, 1051, 1088-1089, 1097-1098, 1112, 1114-1115, 1122-1126, 1129-1133, 1137, 1140-1149, 1153-1160, 1164, 1200, 1208, 1250-1258, 1275-1277, 1280-1285, 1351, 1362-1364, 1370-1371, 1380-1381, 1385-1386, 1388, 1390, 1399, 4899, 4909, 4930, 4933, 4948, 4951-4952, 4960, 4971, 5012, 5039, 5104, 5123, 5142, 5210, 5346-5354, 8482, 8612, 9027, 9030, 9044, 9048-9049, 9056, 9061, 9066, 9238, 12712, 13121, 13218, 13488, 16684, 16804, 17248, 21427, 28709
21427	835, 927, 947, 1200, 1208, 4931, 9027, 13488, 17584
25473	37, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1252, 4386, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 8229, 8482, 9030, 9044, 9049, 9061, 9066, 25479, 25617, 25619, 28709
25479	37, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 836, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1115, 1252, 4909, 4932, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5211, 8229, 9030, 9044, 9049, 9061, 9066, 25473, 25617, 25619, 28709
25480	37, 500, 904, 1114, 5210, 8229, 28709
25508	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25518, 29614, 33698-33700, 37796
25510	933-934, 949, 5029, 5045, 5460, 9125, 13221, 17317, 25525, 29621, 33717, 37813
25512	935-936, 946, 1381, 5031, 5477, 5484, 9127, 13223
25514	937-938, 950, 1370, 5033, 5046, 9142
25518	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 29614, 33698-33700, 37796
25520	933, 944, 949, 5029, 5045, 5460, 9125, 13221, 17317, 25525, 29616, 29621, 33717, 37813
25524	937, 948, 950, 1370, 5033, 5046, 9142, 29620
25525	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 29616, 29621, 33717, 37813
25617	37, 273, 277-278, 280, 284-285, 290, 297, 367, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 895-897, 903, 912, 916, 1025-1027, 1040-1043, 1088, 1252, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 4992, 5012, 5123, 8229, 8482, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 25473, 25479, 25619, 25664, 28709
25619	37, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 1025-1027, 1040-1043, 1088, 1114, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5210, 8229, 8482, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 25473, 25479, 25617, 25664, 28709

Table 102. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
25664	37, 273, 277-278, 280, 284-285, 290, 297, 367, 500, 819, 833, 836, 850, 852, 855, 857, 870-871, 875, 891, 1025-1027, 1040-1043, 1088, 1126, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4971, 5123, 8229, 8482, 9025, 9044, 9049, 13121, 25617, 25619, 28709
28709	37, 256, 273, 275, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 720, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-866, 869-871, 874-875, 880, 897, 903-905, 912, 914-916, 920-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1097, 1100, 1112, 1114-1115, 1122, 1124, 1126, 1130-1132, 1137, 1140-1149, 1200, 1208, 1250-1255, 1257-1258, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5210-5211, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 16804, 17248, 17584, 25473, 25479, 25480, 25617, 25619, 25664
29614	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 33698-33700, 37796
29616	933, 944, 949, 5029, 5045, 5460, 9125, 13221, 17317, 25520, 25525, 29621, 33717, 37813
29620	937, 948, 950, 1370, 5033, 5046, 9142, 25524
29621	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 33717, 37813
33698	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33699-33700, 37796
33699	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698, 33700, 37796
33700	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33699, 37796
33717	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 37813
37796	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700
37813	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 33717

OS/2 conversion support

MQSeries for OS/2 Warp V5, or later, supports conversion between any of the CCSIDS listed below:

037	256	259	273	274	277
278	280	282	284	285	287
290	293	297	300	301	361
363	367	382	383	385	386
387	388	389	391	392	393
394	395	420	423	424	437
500	803	813	819	829	833
834	835	836	837	838	850
851	852	855	856	857	858
860	861	862	863	864	865
866	867	868	869	870	871
874	875	878	880	891	895
896	897	903	904	905	907
909	910	912	913	914	915
916	918	919	920	921	922

OS/2 conversion support

923	924	927	930	932	933
935	937	938 (1)	939	941	942
943	946	947	948	949	950
951	952	954 (2)	955	960	961
963	964	970	971	1004	1006
1008	1009	1010	1011	1012	1013
1014	1015	1016	1017	1018	1019
1025	1026	1027	1028	1038	1040
1041	1042	1043	1046	1047	1050
1051	1088	1089	1092	1097	1098
1112	1114	1115	1116	1117	1118
1119	1122	1123	1124	1129	1130
1132	1133	1140	1141	1142	1143
1144	1145	1146	1147	1148	1149
1200	1208	1250	1251	1252	1253
1254	1255	1256	1257	1258	1275
1276	1277	1350	1363	1364	1380
1381	1382	1383	1386	1388	4899
4948	4951	4952	4960	5026	5035
5037	5039	5048	5049	5050 (2)	5067
5142	5346	5347	5348	5349	5350
5351	5352	5353	5354	5478	8612
9030	9048	9056	9066	9145	12712
13488	17584	28709	33722		

Notes:

1. – 938 uses 948 for conversion.
2. – 954 and 5050 use 33722 for conversion.

OS/400 conversion support

A full list of CCSIDs, and conversions supported by OS/400, can be found in the appropriate AS/400 publication relating to your operating system.

Unicode conversion support

Some platforms support the conversion of user data to or from Unicode encoding. The two forms of unicode encoding supported are UCS-2 (CCSIDs 1200, 13488, and 17584) and UTF-8 (CCSID 1208).

Note: WebSphere MQ does not support UCS-2 queue manager CCSIDs so message header data cannot be encoded in UCS-2.

MQSeries OS/2 support for Unicode

On MQSeries for OS/2 Warp V5 or later, conversion on OS/2 to and from the Unicode CCSIDs is supported for all supported CCSIDs. See “OS/2 conversion support” on page 669

WebSphere MQ AIX support for Unicode

On MQSeries for AIX Version 5 or later, conversion on AIX to and from the Unicode CCSIDs is supported for the following CCSIDs:

037	273	278	280	284	285
297	423	437	500	813	819
850	852	856	857	858	860

Unicode conversion support

861	865	867	869	875	878
880	912	915	916	920	923
924	932	933	935	937	938
939	942	943	948	949	950
954	964	970	1026	1046	1089
1129	1130	1131	1132	1133	1140
1141	1142	1143	1144	1145	1146
1147	1148	1149	1200	1208	1250
1251	1253	1254	1258	1280	1281
1282	1283	1284	1285	1363	1364
1381	1383	1386	1388	4899	5026
5035	5050	5346	5347	5348	5349
5350	5351	5352	5353	5354	5488
9048	12712	13488	17584	33722	

WebSphere MQ HP-UX support for Unicode

On MQSeries for HP-UX Version 5 or later, conversion on HP to, and from, the Unicode CCSIDs is supported for the following CCSIDs:

813	819	874	912	915	916
920	932	938	950	954	964
970	1051	1089	1200	1208	1381
5050	5488	13488	33722		

Note: HP-UX version 10 does not support conversion into or from UTF-8

WebSphere MQ (for Windows, Solaris, and Linux) and MQSeries (for Compaq NSK and Tru64) support for Unicode

On WebSphere MQ for Windows, WebSphere MQ for Solaris, WebSphere MQ for Linux and MQSeries for Compaq Tru64 UNIX conversion to, and from, the Unicode CCSIDs is supported for the following CCSIDs:

037	277	278	280	284	285
290	297	300	301	420	424
437	500	813	819	833	835
836	837	838	850	852	855
856	857	858	860	861	862
863	864	865	866	867	868
869	870	871	874	875	878
880	891	897	903	904	912
913 (5)	915	916	918	920	921
922	923	924	927	928	930
931 (1)	932 (2)	933	935	937	938 (3)
939	941	942	943	947	948
949	950	951	954 (4)	964	970
1006	1025	1026	1027	1040	1041
1042	1043	1046	1047	1051	1088
1089	1097	1098	1112	1114	1115
1122	1123	1124	1129	1130	1132
1133	1140	1141	1142	1143	1144
1145	1146	1147	1148	1149	1200
1208	1250	1251	1252	1253	1254
1255	1256	1257	1258	1275	1280

Unicode conversion support

1281	1282	1283	1363	1364	1380
1381	1383	1386	1388	4899	5050
5346	5347	5348	5349	5350	5351
5352	5353	5354	5488 (5)	9048	12712
13488	17584	33722 (4)			

Notes:

1. – 931 uses 939 for conversion.
2. – 932 uses 942 for conversion.
3. – 938 uses 948 for conversion.
4. – 954 and 33722 use 5050 for conversion.
5. – On Windows, Linux and Solaris only.

OS/400 support for Unicode

OS/400 supports a special variant of UNICODE with CCSID 61952 from Version 3.1 onwards. Version 3.7 and later versions also support UNICODE CCSID 13488. Version 4.3 and later versions support the UTF-8 UNICODE CCSID 1208. For details on UNICODE support refer to the appropriate AS/400 publication relating to your operating system.

WebSphere MQ for z/OS support for Unicode

On WebSphere MQ for z/OS conversion to and from the Unicode CCSIDs is supported for the following CCSIDs:

37	256	259	273	275	277
278	280	282	284	285	290
293	297	300	301	367	420
423	424	437	500	720	737
775	803	806	808	813	819
833	834	835	836	837	838
848	849	850	851	852	855
856	857	858	859	860	861
862	863	864	865	866	867
868	869	870	871	872	874
875	878	880	891	895	896
897	901	902	903	904	905
912	914	915	916	918	920
921	922	923	924	927	928
930	932	933	935	937	939
941	942	943	944	946	947
948	949	950	951	1004	1006
1008	1009	1010	1011	1012	1013
1014	1015	1016	1017	1018	1019
1025	1026	1027	1040	1041	1042
1043	1046	1047	1051	1088	1089
1097	1098	1112	1114	1115	1122
1123	1124	1125	1126	1129	1130
1131	1132	1133	1137	1140	1141
1142	1143	1144	1145	1146	1147
1148	1149	1153	1154	1155	1156
1157	1158	1159	1160	1161	1162
1164	1200	1208	1250	1251	1252
1253	1254	1255	1256	1257	1258
1275	1276	1277	1280	1281	1282
1283	1284	1285	1351	1362	1363

Unicode conversion support

1364	1370	1371	1380	1381	1385
1386	1388	1390	1399	4899	4909
4930	4933	4948	4951	4952	4960
4971	5012	5039	5104	5123	5142
5210	5346	5347	5348	5349	5350
5351	5352	5353	5354	5488	8482
8612	9027	9030	9044	9048	9049
9056	9061	9066	9238	12712	13121
13218	13488	16684	16804	17248	17584
21427	28709				

Unicode conversion support

Appendix I. Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Notices

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AD/Cycle	AIX	AS/400
CICS	DB2	IBM
IBMLink	IMS	iSeries
MQSeries	MVS/ESA	OpenEdition
OS/2	OS/390	OS/400
RACF	SAA	System/390
VisualAge	VM/ESA	VSE/ESA
WebSphere	z/OS	zSeries

Lotus Notes is a trademark of Lotus Development Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

ActionMedia, LANDesk, MMX, Pentium, and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Object attributes

Index

A

AbendCode field 43
AccountingToken field
 MQMD structure 144
 MQPMR structure 252
ADSDescriptor field 43
aliasing
 queue manager 457
 reply queue 457
AlterationDate attribute
 authentication information 521
 namelist 491
 process definition 495
 queue 460
 queue manager 502
AlterationTime attribute
 authentication information 521
 namelist 492
 process definition 496
 queue 461
 queue manager 503
AlternateSecurityId field 212
AlternateUserId field 213
AppId
 attribute 496
 field
 MQTM structure 293
 MQTMC2 structure 300
AppIdentityData field 146
AppOriginData field 146
AppType
 attribute 496
 field
 MQTM structure 293
 MQTMC2 structure 300
AppOptions field 590
assembler language
 assemblers supported xix
AttentionId field 44
attributes
 authentication information 521
 namelist 491
 process definition 495
 queue 457
 queue manager 501
authentication information
 attributes 521
 authentication information record 31
Authenticator field 44, 134
AuthInfoConnName
 attribute 522
AuthInfoConnName field
 MQAIR structure 31
AuthInfoDesc
 attribute 522
AuthInfoName
 attribute 522
AuthInfoRecCount field
 MQSCO structure 285
AuthInfoRecOffset field
 MQSCO structure 286

AuthInfoRecPtr field
 MQSCO structure 286
AuthInfoType
 attribute 522
AuthInfoType field
 MQAIR structure 32
AuthorityEvent attribute 503

B

BackoutCount field 147
BackoutRequeueQName attribute 461
BackoutThreshold attribute 461
BaseQName attribute 462
Basic
 compilers supported xix
begin options structure 37
BeginOptions parameter 341
Buffer parameter
 declaring 333
 MQGET call 380
 MQPUT call 425
 MQPUT1 call 438
BufferLength parameter
 MQGET call 380
 MQPUT call 424
 MQPUT1 call 438
built-in formats 157

C

C language
 compilers supported xix
C programming language
 data types 20
 functions 19
 header files 19
 initial values for dynamic
 structures 21
 initial values for structures 21
 manipulating binary strings 20
 manipulating character strings 20
 notational conventions 22
 parameters with undefined data
 types 20
 use from C++ 22
 using calls 333
C++ language
 compilers supported xix
calls
 conventions used 331
 detailed description
 MQ_DATA_CONV_EXIT 602
 MQBACK 335
 MQBEGIN 341
 MQCLOSE 345
 MQCMIT 353
 MQCONN 359
 MQCONNX 369
 MQDISC 373

calls (*continued*)
 detailed description (*continued*)
 MQGET 379
 MQINQ 393
 MQOPEN 405
 MQPUT 423
 MQPUT1 437
 MQSET 447
 MQXCNCV 595
CancelCode field 44
CCSID language support 611
CFStrucName attribute 462
ChannelAutoDef attribute 503
ChannelAutoDefEvent attribute 504
ChannelAutoDefExit attribute 504
CharAttrLength parameter
 MQINQ call 398
 MQSET call 449
CharAttrs parameter
 MQINQ call 398
 MQSET call 449
ClientConnOffset field 62
ClientConnPtr field 62
ClusterName attribute 462
ClusterNamelist attribute 463
ClusterWorkloadData attribute 505
ClusterWorkloadExit attribute 505
ClusterWorkloadLength attribute 505
CMQB Visual Basic header file 30
COBOL
 compilers supported xix
COBOL programming language
 COPY files 22
 named constants 24
 notational conventions 25
 pointer data type 24
 structures 23
code-page conversions 611
coded character set identifier 506
CodedCharSetId
 attribute 506
 field
 MQCIH structure 45
 MQDH structure 76
 MQDLH structure 85
 MQDXP structure 590
 MQIIH structure 134
 MQMD structure 147
 MQMDE structure 205
 MQRFH structure 255
 MQRFH2 structure 262
 MQRMH structure 273
 MQWIH structure 306
Codepage support 614
 Arabic 634
 Cyrillic 626
 Danish and Norwegian 616
 Eastern European languages 625
 Estonian 627
 Farsi 635
 Finnish and Swedish 617

- Codepage support (*continued*)
 - French 621
 - German 615
 - Greek 630
 - Hebrew 632
 - Icelandic 624
 - Italian 618
 - Japanese Kanji/ Katakana Mixed 646
 - Japanese Kanji/ Latin Mixed 644
 - Japanese Katakana SBCS 642
 - Japanese Latin SBCS 640
 - Korean 648
 - Lao 638
 - Latvian and Lithuanian 628
 - Multilingual 622
 - Portuguese 623
 - Simplified Chinese 649
 - Spanish 619
 - Thai 637
 - Traditional Chinese 651
 - Turkish 631
 - UK English / Gaelic 620
 - Ukranian 629
 - Urdu 636
 - US English 614
 - Vietnamese 639
- CommandInputQName attribute 506
- CommandLevel attribute 506
- CommitMode field 134
- Compaq NonStop Kernel
 - IPC message 609
 - signal notification 609
- compatibility mode 365
- CompCode field
 - MQCIH structure 45
 - MQDXP structure 590
 - MQRR structure 283
- CompCode parameter
 - MQBACK call 335
 - MQBEGIN call 341
 - MQCLOSE call 347
 - MQCMIT call 353
 - MQCONN call 362
 - MQCONNX call 369
 - MQDISC call 373
 - MQGET call 381
 - MQINQ call 398
 - MQOPEN call 412
 - MQPUT call 425
 - MQPUT1 call 438
 - MQSET call 449
 - MQXCNVC call 599
- compilers supported xix
- completion code 527
- connect options structure 61
- ConnectOpts parameter 369
- ConnTag field 64
- constants, values of 529
 - accounting token (MQACT_*) 530
 - accounting token type (MQACTT_*) 531
 - application type (MQAT_*) 531
 - authentication information record structure identifier (MQAIR_*) 531
 - authentication information record version (MQAIR_*) 531
 - constants, values of (*continued*)
 - authentication information type (MQAIT_*) 531
 - backout hardening (MQQA_*) 558
 - begin options (MQBO_*) 532
 - begin options structure identifier (MQBO_*) 532
 - begin options version (MQBO_*) 533
 - binding (MQBND_*) 532
 - channel auto-definition (MQCHAD_*) 535
 - character attribute selectors (MQCA_*) 533
 - CICS bridge return code (MQCRC_*) 538
 - CICS function name (MQCFUNC_*) 535
 - CICS header ADS descriptor (MQCADSD_*) 534
 - CICS header conversational task (MQCCT_*) 534
 - CICS header facility (MQCFAC_*) 535
 - CICS header flags (MQCIH_*) 536
 - CICS header get-wait interval (MQCGWI_*) 535
 - CICS header length (MQCIH_*) 536
 - CICS header link type (MQCLT_*) 537
 - CICS header output data length (MQCODL_*) 538
 - CICS header structure identifier (MQCIH_*) 536
 - CICS header task end status (MQCTES_*) 539
 - CICS header transaction start code (MQCSC_*) 539
 - CICS header unit-of-work control (MQCUOWC_*) 539
 - CICS header version (MQCIH_*) 537
 - close options (MQCO_*) 538
 - coded character set identifier (MQCCSI_*) 534
 - command level (MQCMDL_*) 537
 - completion codes (MQCC_*) 534
 - connect options (MQCNO_*) 537
 - connect options structure identifier (MQCNO_*) 538
 - connect options version (MQCNO_*) 538
 - connection handle (MQHC_*) 547
 - connection tag (MQCT_*) 539
 - convert-characters masks and factors (MQDCC_*) 540
 - convert-characters options (MQDCC_*) 540
 - correlation identifier (MQCL_*) 536
 - data-conversion-exit parameter structure identifier (MQDXP_*) 541
 - data-conversion-exit parameter structure version (MQDXP_*) 542
 - data-conversion-exit response (MQXDR_*) 566
 - dead-letter header structure identifier (MQDLH_*) 541
 - dead-letter header version (MQDLH_*) 541
 - distribution header flags (MQDHF_*) 541
 - distribution header structure identifier (MQDH_*) 540
 - distribution header version (MQDH_*) 540
 - distribution list support (MQDL_*) 541
 - encoding (MQENC_*) 542
 - encoding for binary integers (MQENC_*) 543
 - encoding for floating-point numbers (MQENC_*) 543
 - encoding for packed-decimal integers (MQENC_*) 543
 - encoding masks (MQENC_*) 542
 - event reporting (MQEVR_*) 543
 - event reporting (MQQSIE_*) 559
 - exit command identifier (MQXC_*) 565
 - exit identifier (MQXT_*) 567
 - exit parameter block structure identifier (MQXP_*) 566
 - exit parameter block version (MQXP_*) 566
 - exit reason (MQXR_*) 567
 - exit response (MQXCC_*) 565
 - exit user area (MQXUA_*) 567
 - expiry interval (MQEI_*) 542
 - expiry scan interval (MQEXPI_*) 543
 - feedback (MQFB_*) 544
 - format (MQFMT_*) 545
 - get message options (MQGMO_*) 546
 - get message options structure identifier (MQGMO_*) 546
 - get message options version (MQGMO_*) 546
 - group identifier (MQGI_*) 545
 - group status (MQGS_*) 547
 - IMS authenticator (MQIAUT_*) 548
 - IMS commit mode (MQICM_*) 549
 - IMS header flags (MQIIH_*) 549
 - IMS header length (MQIIH_*) 550
 - IMS header structure identifier (MQIIH_*) 550
 - IMS header version (MQIIH_*) 550
 - IMS security scope (MQISS_*) 550
 - IMS transaction instance identifier (MQITII_*) 551
 - IMS transaction state (MQITS_*) 551
 - Index type (MQIT_*) 550
 - inhibit get (MQQA_*) 558
 - inhibit put (MQQA_*) 558
 - integer attribute selectors (MQIA_*) 547
 - integer attribute value (MQIAV_*) 549
 - intra-group queuing (MQIGQ_*) 549
 - intra-group queuing put authority (MQIGQPA_*) 549
 - lengths of character string and byte fields (MQ_*) 529
 - match options (MQMO_*) 553
 - message delivery sequence (MQMDS_*) 552

constants, values of (*continued*)

- message descriptor extension flags (MQMDEF_*) 552
- message descriptor extension length (MQMDE_*) 551
- message descriptor extension structure identifier (MQMDE_*) 552
- message descriptor extension version (MQMDE_*) 552
- message descriptor structure identifier (MQMD_*) 551
- message descriptor version (MQMD_*) 551
- message flags (MQMF_*) 552
- message identifier (MQML_*) 553
- message token (MQMTOK_*) 554
- message type (MQMT_*) 553
- message-flags masks (MQMF_*) 553
- name count (MQNC_*) 554
- namelist type (MQNT_*) 554
- object descriptor length (MQOD_*) 554
- object descriptor structure identifier (MQOD_*) 554
- object descriptor version (MQOD_*) 554
- object handle (MQHO_*) 547
- object instance identifier (MQOIL_*) 555
- object type (MQOT_*) 556
- open options (MQOO_*) 555
- original length (MQOL_*) 555
- persistence (MQPER_*) 556
- platform (MQPL_*) 556
- priority (MQPRI_*) 557
- put message options (MQPMO_*) 556
- put message options length (MQPMO_*) 557
- put message options structure identifier (MQPMO_*) 557
- put message options version (MQPMO_*) 557
- put message record field flags (MQPMRF_*) 557
- queue definition type (MQQDT_*) 558
- queue shareability (MQQA_*) 558
- queue type (MQQT_*) 559
- queue-sharing group disposition (MQQSGD_*) 558
- reason codes (MQRC_*) 559
- reference message header flags (MQRMHF_*) 560
- reference message header structure identifier (MQRMH_*) 560
- reference message header version (MQRMH_*) 560
- report options (MQRO_*) 561
- report-options masks (MQRO_*) 561
- returned length (MQRL_*) 560
- rules and formatting header flags (MQRFH_*) 559
- rules and formatting header length (MQRFH_*) 559
- rules and formatting header structure identifier (MQRFH_*) 559

constants, values of (*continued*)

- rules and formatting header version (MQRFH_*) 560
- scope (MQSCO_*) 561
- security identifier (MQSID_*) 562
- security identifier type (MQSIDT_*) 562
- segment status (MQSS_*) 563
- segmentation (MQSEG_*) 562
- signal event-control-block completion codes (MQEC_*) 542
- SSL configuration options structure identifier (MQSCO_*) 561
- SSL configuration options version (MQSCO_*) 562
- syncpoint (MQSP_*) 562
- transmission queue header structure identifier 566
- transmission queue header version (MQXQH_*) 566
- trigger controls (MQTC_*) 563
- trigger message (character format) structure identifier (MQTMC_*) 563
- trigger message (character format) version (MQTMC_*) 564
- trigger message structure identifier (MQTM_*) 563
- trigger message version (MQTM_*) 563
- trigger type (MQTT_*) 564
- undelivered-message header structure identifier (MQDLH_*) 541
- undelivered-message header version (MQDLH_*) 541
- usage (MQUS_*) 564
- wait interval (MQWI_*) 564
- workload information header flags (MQWIH_*) 564
- workload information header structure identifier (MQWIH_*) 565
- workload information header structure length (MQWIH_*) 565
- workload information header version (MQWIH_*) 565

Context field 230

ConversationalTask field 45

conversion of report messages 587

conversions, code-page 611

COPY files – COBOL programming language 22

CorrelId field

- MQMD structure 149
- MQPMR structure 252

CreationDate attribute 463

CreationTime attribute 464

CryptoHardware field

- MQSCO structure 287

CurrentQDepth attribute 464

CursorPosition field 45

D

data conversion

- processing conventions 583
- report messages 587

data types, conventions used 18

data types, detailed description

elementary

- assembler language 14
- C programming language 10
- COBOL programming language 13
- MQBYTE 8
- MQBYTEn 8
- MQCHAR 8
- MQCHARn 9
- MQHCONN 9
- MQHOBj 9
- MQLONG 9
- MQPID 9
- MQPTR 9
- MQTID 10
- overview 7
- PL/I language 13
- PMQCHAR 10
- PMQLONG 10
- PMQMD 10
- TAL programming language 15
- Visual Basic 15

structure

- MQAIR 31
- MQBO 37
- MQCIH 41
- MQCNO 61
- MQDH 75
- MQDLH 83
- MQDXP 589
- MQGMO 95
- MQIIH 133
- MQMD 141
- MQMDE 203
- MQOD 211
- MQOR 227
- MQPMO 229
- MQPMR 251
- MQRFH 255
- MQRFH2 261
- MQRMH 271
- MQRR 283
- MQSCO 285
- MQTM 291
- MQTMC2 299
- MQWIH 305
- MQXP 311
- MQXQH 317

programming considerations 17

rules 18

DataConvExitParms parameter 602

DataLength

- field, MQDXP structure 590
- parameter
- MQGET call 381
- MQXCNCV call 599

DataLogicalLength field 273

DataLogicalOffset field 273

DataLogicalOffset2 field 274

dead-letter header structure 83

DeadLetterQName attribute 509

DefBind attribute 465

DefinitionType attribute 465

DefInputOpenOption attribute 466

DefPersistence attribute 467

DefPriority attribute 468

- DefXmitQName attribute 510
- DestEnvLength field 274
- DestEnvOffset field 274
- DestNameLength field 275
- DestNameOffset field 275
- DestQMGrName field 85
- DestQName field 86
- DistLists attribute 468, 510
- distribution header structure 75
- distribution lists 468, 510
- dynamic queue 405
- DynamicQName field 214

E

- Encoding field
 - MQCIH structure 45
 - MQDHL structure 77
 - MQDLH structure 86
 - MQDXP structure 591
 - MQIIH structure 135
 - MQMD structure 150
 - MQMDE structure 206
 - MQRFH structure 256
 - MQRFH2 structure 262
 - MQRMH structure 275
 - MQWIH structure 306
 - using 571
- EnvData
 - attribute 497
 - field
 - MQTM structure 294
 - MQTMC2 structure 300
- environment variable –
 - MQ_CONNECT_TYPE 66
- ErrorOffset field 45
- exit parameter block 311
- ExitCommand field 311
- ExitId field 312
- ExitOptions field 591
- ExitParmCount field 312
- ExitReason field 312
- ExitResponse field
 - MQDXP structure 591
 - MQXP structure 313
- ExitUserArea field 313
- expired-message processing 510
- Expiry field 151
- ExpiryInterval attribute 510

F

- Facility field 46
- FacilityKeepTime field 46
- FacilityLike field 46
- Feedback field
 - MQMD structure 153
 - MQPMR structure 252
- Flags field
 - MQCIH structure 46
 - MQDHL structure 77
 - MQIIH structure 135
 - MQMDE structure 206
 - MQRFH structure 256
 - MQRFH2 structure 263
 - MQRMH structure 275

- Flags field (*continued*)
 - MQWIH structure 306
- fonts in this book xxii
- form files (.BAS) 30
- Format field
 - MQCIH structure 47
 - MQDHL structure 78
 - MQDLH structure 86
 - MQIIH structure 135
 - MQMD structure 157
 - MQMDE structure 206
 - MQRFH structure 256
 - MQRFH2 structure 263
 - MQRMH structure 276
 - MQWIH structure 306
- formats built-in 157
- Function field 47
- functions – C programming language 19

G

- get-message options structure 95
- GetMsgOpts parameter 380
- GetWaitInterval field 48
- GroupId field
 - MQMD structure 164
 - MQMDE structure 207
 - MQPMR structure 253
- GroupStatus field 96

H

- handle scope 362
- handle sharing 68
- handles 513
- HardenGetBackout attribute 469
- Hconn field 592
- Hconn parameter
 - MQBACK call 335
 - MQBEGIN call 341
 - MQCLOSE call 345
 - MQCMIT call 353
 - MQCONN call 362
 - MQCONNXX call 369
 - MQDISC call 373
 - MQGET call 379
 - MQINQ call 393
 - MQOPEN call 405
 - MQPUT call 423
 - MQPUT1 call 437
 - MQSET call 447
 - MQXCNCV call 596
- scope 362
- header files
 - C programming language 19
 - Visual Basic programming language 30
- Hobj parameter
 - MQCLOSE call 345
 - MQGET call 379
 - MQINQ call 393
 - MQOPEN call 412
 - MQPUT call 423
 - MQSET call 447

I

- IGQPutAuthority attribute 511
- IGQUserId attribute 512
- InBuffer parameter 603
- InBufferLength parameter 603
- INCLUDE files – PL/I programming language 25
- IndexType attribute 470
- InhibitEvent attribute 512
- InhibitGet attribute 472
- InhibitPut attribute 473
- InitiationQName attribute 473
- InputItem field 48
- IntAttrCount parameter
 - MQINQ call 397
 - MQSET call 448
- IntAttrs parameter
 - MQINQ call 397
 - MQSET call 449
- intra-group queuing 511, 512
- IntraGroupQueuing attribute 512
- InvalidDestCount field
 - MQOD structure 214
 - MQPMO structure 230
- IPC message (Compaq NonStop Kernel only) 609

K

- KeyRepository field
 - MQSCO structure 287
- KnownDestCount field 215, 230

L

- languages supported xix
- LDAPPassword
 - attribute 522
- LDAPPassword field
 - MQAIR structure 32
- LDAPUserName
 - attribute 522
- LDAPUserNameLength field
 - MQAIR structure 32
- LDAPUserNameOffset field
 - MQAIR structure 32
- LDAPUserNamePtr field
 - MQAIR structure 33
- LinkType field 48
- LocalEvent attribute 513
- LTermOverride field 135

M

- macros 26
- MatchOptions field 96
- MaxHandles attribute 513
- MaxMsgLength attribute
 - queue 474
 - queue manager 514
- MaxPriority attribute 514
- MaxQDepth attribute 475
- MaxUncommittedMsgs attribute 515
- message descriptor extension
 - structure 203

message descriptor structure 141
 message order 386, 430, 444
 MFSSMapName field 136
 MQ_* values 529
 MQ_CONNECT_TYPE environment variable 66
 MQ_DATA_CONV_EXIT call 602
 MQACT_* values 145
 MQAIR structure 31
 MQAIR_* values 33
 MQAIR_DEFAULT 34
 MQAT_* values
 AppType attribute 497
 field 293
 PutAppType field 178
 MQBACK call 335
 MQBEGIN call 341
 MQBND_* values 465
 MQBO structure 37
 MQBO_* values 37
 MQBO_DEFAULT 38
 MQBYTE 8
 MQBYTEn 8
 MQCA_* values 394, 448
 MQCC_* values 527
 MQCCSI_* values 147
 MQCD_CLIENT_CONN_DEFAULT 64
 MQCD_DEFAULT 64
 MQCFUNC_* values 47
 MQCGWI_* values 48
 MQCHAR 8
 MQCHARn 9
 MQCI_* values 149
 MQCIH structure 41
 MQCIH_* values 52
 MQCIH_DEFAULT 55
 MQCLOSE call 345
 MQCLT_* values 48
 MQCMDL_* values 506
 MQCMIT call 353
 MQCNO structure 61
 MQCNO_* values 65, 70
 MQCNO_DEFAULT 71
 MQCNOCD structure 64
 MQCO_* values 346
 MQCODL_* values 49
 MQCONN call 359
 MQCONNX call 369
 MQCONNXAny call 64, 371
 MQCRC_* values 50
 MQCT_* values 65
 MQCUOWC_* values 53
 MQDCC_* values 596
 MQDH structure 75
 MQDH_* values 79
 MQDH_DEFAULT 80
 MQDHF_* values 77
 MQDISC call 373
 MQDL_* values 469, 510
 MQDLH structure 83
 MQDLH_* values 89
 MQDLH_DEFAULT 90
 MQDXP structure 589
 MQDXP_* values 594
 MQEC_* values 125
 MQEI_* values 153
 MQENC_* values 150
 MQEVR_* values
 AuthorityEvent attribute 503
 ChannelAutoDefEvent attribute 504
 InhibitEvent attribute 512
 LocalEvent attribute 513
 PerformanceEvent attribute 515
 QDepthHighEvent attribute 478
 QDepthLowEvent attribute 478
 QDepthMaxEvent attribute 479
 RemoteEvent attribute 517
 StartStopEvent attribute 519
 MQEXPI_* values 510
 MQFB_* values 88, 153
 MQFMT_* values 157
 MQGET call 379
 MQGETAny call 389
 MQGL_* values 165
 MQGMO structure 95
 MQGMO_* values 100, 126
 MQGMO_DEFAULT 128
 MQGS_* values 96
 MQHC_* values 373
 MQHCONN 9
 MQHO_* values 345
 MQHOBJ 9
 MQIA_* values 394, 448
 MQIAUT_* values 134
 MQIAV_* values 398
 MQICM_* values 134
 MQIGQ_* values 512
 MQIGQPA_* values 511
 MQIIH structure 133
 MQIIH_* values 136
 MQIIH_DEFAULT 138
 MQINQ call 393
 MQISS_* values 136
 MQIT_* values 470
 MQITIL_* values 137
 MQITS_* values 137
 MQLONG 9
 MQMD structure 141
 MQMD_* values 194, 196
 MQMD_DEFAULT 197
 MQMDE structure 203
 MQMDE_* values 207
 MQMDE_DEFAULT 208
 MQMDEF_* values 206
 MQMDS_* values 475
 MQMF_* values 165
 MQMI_* values 171
 MQMO_* values 97
 MQMT_* values 172
 MQMTOK_* values 99
 MQNC_* values 492
 MQNT_* values 493
 MQOD structure 211
 MQOD_* values 221
 MQOD_DEFAULT 222
 MQOII_* values 276
 MQOL_* values 174
 MQOO_* values 406, 466
 MQOPEN call 405
 MQOR structure 227
 MQOR_DEFAULT 228
 MQOT_* values 218
 MQPER_* values 175
 MQPID 9
 MQPL_* values 516
 MQPMO structure 229
 MQPMO_* values 231, 245
 MQPMO_DEFAULT 246
 MQPMR structure 251
 MQPMRF_* values 78, 240
 MQPRI_* values 176
 MQPTR 9
 MQPUT call 423
 MQPUT1 call 437
 MQPUT1Any call 444
 MQPUTAny call 434
 MQQA_* values
 InhibitGet attribute 472
 InhibitPut attribute 473
 Shareability attribute 485
 MQQDT_* values 465
 MQQSGD_* values 482, 493, 499
 MQQSIE_* values 481
 MQQT_* values 462, 482
 MQRC_* values 157
 MQRFH structure 255
 MQRFH_* values 257, 266
 MQRFH_DEFAULT 258
 MQRFH2 structure 261
 MQRFH2_DEFAULT 267
 MQRL_* values 124
 MQRMH structure 271
 MQRMH_* values 277, 278
 MQRMH_DEFAULT 279
 MQRMHF_* values 275
 MQRO_* values 184
 MQRR structure 283
 MQRR_DEFAULT 284
 MQSCO structure 285
 MQSCO_* values 288, 484
 MQSCO_DEFAULT 289
 MQSEG_* values 124
 MQSeries for AT&T GIS UNIX compilers supported xix
 MQSeries for Compaq NonStop Kernel compilers supported xix
 MQSeries for Compaq OpenVMS Alpha compilers supported xix
 MQSeries for DIGITAL UNIX (Compaq Tru64 UNIX) compilers supported xix
 MQSeries for OS/2 Warp compilers supported xix
 MQSeries for SINIX and DC/OSx compilers supported xix
 MQSeries for VSE/ESA compilers supported xix
 MQSET call 447
 MQSID_* values 213
 MQSIDT_* values 212
 MQSP_* values 519
 MQSS_* values 125
 MQTC_* values 486
 MQTID 10
 MQTM structure 291
 MQTM_* values 295
 MQTM_DEFAULT 297
 MQTMC_* values 300, 301
 MQTMC2 structure 299
 MQTMC2_DEFAULT 301

- MQTT_* values 487
- MQUS_* values 488
- MQWI_* values 127
- MQWIH structure 305
- MQWIH_* values 307
- MQWIH_DEFAULT 308
- MQXC_* values 311
- MQXCC_* values 313
- MQXCNCV call 595
- MQXDR_* values 591
- MQXP structure 311
- MQXP_* values 314
- MQXQH structure 317
- MQXQH_* values 321
- MQXQH_DEFAULT 322
- MQXR_* values 312
- MQXT_* values 312
- MQXUA_* values 313
- MsgDeliverySequence attribute 475
- MsgDesc field 320
- MsgDesc parameter
 - MQ_DATA_CONV_EXIT call 602
 - MQGET call 379
 - MQPUT call 423
 - MQPUT1 call 437
- MsgFlags field
 - MQMD structure 165
 - MQMDE structure 207
- MsgId field
 - MQMD structure 170
 - MQPMR structure 253
- MsgSeqNumber field
 - MQMD structure 172
 - MQMDE structure 207
- MsgToken field 99, 307
- MsgType field 172

N

- NameCount attribute 492
- named constants – COBOL programming language 24
- namelist attributes 491
- NamelistDesc attribute 492
- NamelistName attribute 492
- NamelistType attribute 493
- Names attribute 493
- NameValueCCSID field 263
- NameValueData field 264
- NameValueLength field 266
- NameValueString field 257
- NextTransactionId field 49
- notational conventions
 - C programming language 22
 - COBOL programming language 25
 - PL/I programming language 26
 - S/370 assembler programming language 29
 - Visual Basic programming language 30

O

- ObjDesc parameter
 - MQOPEN call 405
 - MQPUT1 call 437

- object descriptor structure 211
- object record structure 227
- ObjectInstanceId field 276
- ObjectName field
 - MQOD structure 215
 - MQOR structure 227
- ObjectQMgrName field
 - MQOD structure 216
 - MQOR structure 228
- ObjectRecOffset field
 - MQDH structure 78
 - MQOD structure 217
- ObjectRecPtr field 218
- ObjectType field
 - MQOD structure 218
 - MQRMH structure 276
- Offset field
 - MQMD structure 173
 - MQMDE structure 207
- OpenInputCount attribute 476
- OpenOutputCount attribute 477
- Options field
 - MQBO structure 37
 - MQCNO structure 65
 - MQGMO structure 100
 - MQPMO structure 231
- Options parameter
 - MQCLOSE call 345
 - MQOPEN call 406
 - MQXCNCV call 596
- ordering of messages 386, 430, 444
- OriginalLength field
 - MQMD structure 174
 - MQMDE structure 207
- OutBuffer parameter 603
- OutBufferLength parameter 603
- OutputDataLength field 49

P

- parameters with undefined data types 20
- PerformanceEvent attribute 515
- persistence 467
- Persistence field 175
- PL/I
 - compilers supported xix
- PL/I programming language
 - INCLUDE files 25
 - notational conventions 26
 - structures 26
- Platform attribute 516
- PMQCHAR 10
- PMQLONG 10
- PMQMD 10
- PMQVOID 333
- pointer data type – COBOL programming language 24
- Priority field 176
- process definition attributes 495
- ProcessDesc attribute 498
- ProcessName
 - attribute
 - process definition 498
 - queue 477
 - field
 - MQTM structure 295

- ProcessName (*continued*)
 - field (*continued*)
 - MQTMC2 structure 300
- put message record structure 251
- put-message options structure 229
- PutApplName field
 - MQDLH structure 86
 - MQMD structure 177
- PutApplType field
 - MQDLH structure 87
 - MQMD structure 178
- PutDate field
 - MQDLH structure 87
 - MQMD structure 181
- PutMsgOpts parameter
 - MQPUT call 424
 - MQPUT1 call 438
- PutMsgRecFields field
 - MQDH structure 78
 - MQPMO structure 240
- PutMsgRecOffset field
 - MQDH structure 79
 - MQPMO structure 241
- PutMsgRecPtr field 242
- PutTime field
 - MQDLH structure 87
 - MQMD structure 181

Q

- QDepthHighEvent attribute 477
- QDepthHighLimit attribute 478
- QDepthLowEvent attribute 478
- QDepthLowLimit attribute 479
- QDepthMaxEvent attribute 479
- QDesc attribute 480
- QMgrDesc attribute 516
- QMgrIdentifier attribute 517
- QMgrName
 - attribute 517
 - field 300
- QMgrName parameter
 - MQCONN call 359
 - MQCONNX call 369
- QName
 - attribute 480
 - field
 - MQTM structure 295
 - MQTMC2 structure 300
- QServiceInterval attribute 480
- QServiceIntervalEvent attribute 481
- QSGDisp attribute
 - namelist 493, 499
 - queue 482
- QSGName attribute 517
- QType attribute 482
- queue attributes 457
- queue manager attributes 501
- queue-manager aliasing 457
- queue-sharing group 216, 360, 517
- queue, dynamic 405

R

- reason codes
 - alphabetic list 528

- Reason field 49
 - MQDLH structure 88
 - MQDXP structure 592
 - MQRR structure 284
- Reason parameter
 - MQBACK call 335
 - MQBEGIN call 342
 - MQCLOSE call 347
 - MQCMIT call 354
 - MQCONN call 363
 - MQCONNX call 370
 - MQDISC call 374
 - MQGET call 381
 - MQINQ call 398
 - MQOPEN call 412
 - MQPUT call 425
 - MQPUT1 call 438
 - MQSET call 449
 - MQXCNV call 600
- RecsPresent field
 - MQDH structure 79
 - MQOD structure 218
 - MQPMO structure 242
- reference message header structure 271
- RemoteEvent attribute 517
- RemoteQMgrName
 - attribute 483
 - field 320
- RemoteQName
 - attribute 483
 - field 321
- RemoteSysId field 49
- RemoteTransId field 50
- reply queue aliasing 457
- ReplyToFormat field 50, 136
- ReplyToQ field 182
- ReplyToQMgr field 183
- Report field
 - MQMD structure 184
 - using 575
- report message conversion 587
- RepositoryName attribute 518
- RepositoryNameList attribute 518
- Reserved field
 - MQIIH structure 136
 - MQWIH structure 307
 - MQXP structure 313
- Reserved1 field 50, 123
- Reserved2 field 50
- Reserved3 field 50
- Reserved4 field 50
- ResolvedQMgrName field
 - MQOD structure 219
 - MQPMO structure 243
- ResolvedQName field
 - MQGMO structure 123
 - MQOD structure 219
 - MQPMO structure 243
- response record structure 283
- ResponseRecOffset field
 - MQOD structure 220
 - MQPMO structure 243
- ResponseRecPtr field
 - MQOD structure 220
 - MQPMO structure 244
- RetentionInterval attribute 484
- return codes 527

- ReturnCode field 50
- ReturnedLength field 124
- RPG language
 - compilers supported xix
 - rules and formatting header structure 255
 - rules and formatting header structure version 2 261

S

- Scope attribute 484
- scope, handles 362
- SecurityScope field 136
- Segmentation field 124
- SegmentStatus field 125
- SelectorCount parameter
 - MQINQ call 393
 - MQSET call 447
- Selectors parameter
 - MQINQ call 394
 - MQSET call 447
- ServiceName field 307
- ServiceStep field 307
- Shareability attribute 485
- shared handles 68
- shared queue 216, 386
- signal notification (Compaq NonStop Kernel only) 609
- Signal1 field 125
- Signal2 field 126
- SourceBuffer parameter 599
- SourceCCSID parameter 599
- SourceLength parameter 599
- SrcEnvLength field 276
- SrcEnvOffset field 276
- SrcNameLength field 277
- SrcNameOffset field 277
- SSL configuration options structure 285
- SSLConfigOffset field 69
- SSLConfigPtr field 69
- StartCode field 51
- StartStopEvent attribute 519
- StorageClass attribute 485
- StrucId field
 - MQAIR structure 33
 - MQBO structure 37
 - MQCIH structure 52
 - MQCNO structure 70
 - MQDH structure 79
 - MQDLH structure 89
 - MQDXP structure 594
 - MQGMO structure 126
 - MQIIH structure 136
 - MQMD structure 194
 - MQMDE structure 207
 - MQOD structure 221
 - MQPMO structure 245
 - MQRFH structure 257
 - MQRFH2 structure 266
 - MQRMH structure 277
 - MQSCO structure 288
 - MQTM structure 295
 - MQTMC2 structure 300
 - MQWIH structure 307
 - MQXP structure 314
 - MQXQH structure 321

- StrucLength field
 - MQCIH structure 52
 - MQDH structure 79
 - MQIIH structure 137
 - MQMDE structure 207
 - MQRFH structure 258
 - MQRFH2 structure 266
 - MQRMH structure 278
 - MQWIH structure 307
- structures – COBOL programming
 - language 23
- structures – PL/I programming
 - language 26
- syncpoint 519
- SyncPoint attribute 519
- System/390 Assembler programming
 - language
 - macros 26
 - notational conventions 29

T

- TAL
 - compilers supported xix
- TargetBuffer parameter 599
- TargetCCSID parameter 599
- TargetLength parameter 599
- TaskEndStatus field 52
- terminology xviii
- Timeout field 245
- TranInstanceId field 137
- TransactionId field 53
- transmission queue header structure 317
- TranState field 137
- trigger message structure 291
- TriggerControl attribute 486
- TriggerData
 - attribute 486
 - field
 - MQTM structure 295
 - MQTMC2 structure 300
- TriggerDepth attribute 487
- triggering 486
- TriggerInterval attribute 519
- TriggerMsgPriority attribute 487
- TriggerType attribute 487
- trusted application 66
- type styles in this book xxii

U

- UCS-2 670
- Uncommitted messages 515
- Unicode 670
- UnknownDestCount field
 - MQOD structure 221
 - MQPMO structure 245
- UOWControl field 53
- Usage attribute 488
- use from C++ 22
- UserData
 - attribute 499
 - field
 - MQTM structure 296
 - MQTMC2 structure 301
- UserIdentifier field 194

V

Version field

- MQAIR structure 34
- MQBO structure 38
- MQCIH structure 54
- MQCNO structure 70
- MQDH structure 80
- MQDLH structure 89
- MQDXP structure 594
- MQGMO structure 127
- MQIIH structure 138
- MQMD structure 196
- MQMDE structure 208
- MQOD structure 221
- MQPMO structure 245
- MQRFH structure 258
- MQRFH2 structure 267
- MQRMH structure 278
- MQSCO structure 288
- MQTM structure 296
- MQTMC2 structure 301
- MQWIH structure 308
- MQXP structure 314
- MQXQH structure 321

Visual Basic programming language

- form files 30
- header files 30
- MQI calls 30
- notational conventions 30
- using data types 29

W

- WaitInterval field 127
- WebSphere MQ for AIX
 - compilers supported xix
- WebSphere MQ for HP-UX
 - compilers supported xix
- WebSphere MQ for iSeries
 - compilers supported xix
- WebSphere MQ for Linux
 - compilers supported xix
- WebSphere MQ for Solaris
 - compilers supported xix
- WebSphere MQ for Windows
 - compilers supported xix
- WebSphere MQ for z/OS
 - compilers and assemblers supported xix
- Windows 95 and Windows 98 clients
 - compilers supported xix
- Windows products xix

X

- XmitQName attribute 488

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To make comments about the functions of IBM products or systems, talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

User Technologies Department (MP095)
IBM United Kingdom Laboratories
Hursley Park
WINCHESTER,
Hampshire
SO21 2JN
United Kingdom

- By fax:
 - From outside the U.K., after your international access code use 44-1962-816151
 - From within the U.K., use 01962-816151
- Electronically, use the appropriate network ID:
 - IBM Mail Exchange: GBIBM2Q9 at 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 U.S.A.

SC34-6062-03

