MQSeries®

Application Programming Reference

SC33-1673-07

MQSeries®

Application Programming Reference

SC33-1673-07

Note!
Before using this information and the product it supports, be sure to read the general information under "Appendix I. Notices" on page 687.

Eighth edition (November 2000)

This edition applies to the following products:

- MQSeries for AIX, V5.1
- MQSeries for AS/400, V5.1
- MQSeries for AT&T GIS UNIX V2.2
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for Compaq (DIGITAL) OpenVMS, V2.2.1.1
- MQSeries for HP-UX, V5.1
- MQSeries for OS/2 Warp, V5.1
- MQSeries for OS/390, V2.2
- MQSeries for SINIX and DC/OSx, V2.2
- MQSeries for Sun Solaris, V5.1
- MQSeries for Tandem NonStop Kernel, V2.2.0.1
- MQSeries for VSE/ESA V2.1
- MQSeries for Windows NT, V5.1
- MQSeries for Windows V2.0
- MQSeries for Windows V2.1

and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 1994, 2000. 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 xvii
Terms used in this book xviii
Language compilers xix
How to use this book xxii
Appearance of text in this book xxii

Summary of changes xxiii
| Changes for this edition (SC33-1673-07) xxiii
Changes for the previous edition included: . . . xxiii
Changes for the sixth edition included: xxiii

Part 1. Data type descriptions 1

Chapter 1. Introduction 7
Elementary data types 7
MQBYTE – Byte 7
MQBYTEn – String of n bytes 7
MQCHAR – Single-byte character 8
MQCHARn – String of n single-byte characters . . 8
MQHCONN – Connection handle 8
MQHOBJ – Object handle 8
MQLONG – Long integer 9
| MQPTR – Pointer 9
C declarations 9
COBOL declarations 10
PL/I declarations (AIX, OS/2, OS/390, VSE/ESA, and Windows NT only) 10
System/390 Assembler declarations (OS/390 only) 11
TAL declarations (Tandem NonStop Kernel only) 12
Visual Basic declarations (Windows 3.1, Windows 95, Windows 98, and Windows NT) 12
Structure data types – introduction 14
Summary 14
Rules for structure data types 14
Conventions used in the descriptions 15
C programming 16
Header files 16
Functions 16
Parameters with undefined data type 17
Data types 17
Manipulating binary strings 17
Manipulating character strings 17
Initial values for structures 18
Initial values for dynamic structures 18
Use from C++ 19
Notational conventions 19
COBOL programming 19
COPY files 19
Structures 20
Pointers 21
Named constants 21
Notational conventions 22
PL/I programming 22
INCLUDE files 22
Structures 22
Named constants 23
Notational conventions 23
System/390 Assembler programming 23
Macros 23
Structures 24
| CMQVERA macro 26
Notational conventions 26
Visual Basic programming 26
Header files 26
Parameters of the MQI calls 27
Initial values for structures 27
Notational conventions 27

Chapter 2. MQBO - Begin options . . . 29
Overview 29
Fields 29
Options (MQLONG) 29
StrucId (MQCHAR4) 29
Version (MQLONG) 30
Initial values and language declarations 30
C declaration 30
COBOL declaration 30
PL/I declaration 31
Visual Basic declaration 31

Chapter 3. MQCIH - CICS information header 33
Overview 34
Fields 35
AbendCode (MQCHAR4) 35
ADSDescriptor (MQLONG) 35
AttentionId (MQCHAR4) 36
Authenticator (MQCHAR8) 36
CancelCode (MQCHAR4) 36
CodedCharSetId (MQLONG) 37
CompCode (MQLONG) 37
ConversationalTask (MQLONG) 37
CursorPosition (MQLONG) 37
Encoding (MQLONG) 37
ErrorOffset (MQLONG) 37
Facility (MQBYTE8) 38
FacilityKeepTime (MQLONG) 38
FacilityLike (MQCHAR4) 38
Flags (MQLONG) 38
Format (MQCHAR8) 39
Function (MQCHAR4) 39
GetWaitInterval (MQLONG) 40
InputItem (MQLONG) 40
LinkType (MQLONG) 40

© Copyright IBM Corp. 1994, 2000

NextTransactionId (MQCHAR4) 40
OutputDataLength (MQLONG) 40
Reason (MQLONG) 41
RemoteSysId (MQCHAR4) 41
RemoteTransId (MQCHAR4) 41
ReplyToFormat (MQCHAR8) 41
Reserved1 (MQCHAR8) 42
Reserved2 (MQCHAR8) 42
Reserved3 (MQCHAR8) 42
Reserved4 (MQLONG) 42
ReturnCode (MQLONG) 42
StartCode (MQCHAR4) 43
StrucId (MQCHAR4) 43
StrucLength (MQLONG) 44
TaskEndStatus (MQLONG) 44
TransactionId (MQCHAR4) 44
UOWControl (MQLONG) 45
Version (MQLONG) 45
Initial values and language declarations 46
C declaration 47
COBOL declaration 48
PL/I declaration 49
System/390 assembler declaration 50

Chapter 4. MQCNO - Connect options 51
Overview 51
Fields 51
ClientConnOffset (MQLONG) 52
ClientConnPtr (MQPTR) 52
ConnTag (MQBYTE128) 54
Options (MQLONG) 54
StrucId (MQCHAR4) 57
Version (MQLONG) 57
Initial values and language declarations 58
C declaration 58
COBOL declaration 58
PL/I declaration 59
System/390 assembler declaration (OS/390) . . 59
Visual Basic declaration 59

Chapter 5. MQDH - Distribution header 61
Overview 61
Fields 62
CodedCharSetId (MQLONG) 62
Encoding (MQLONG) 63
Flags (MQLONG) 63
Format (MQCHAR8) 64
ObjectRecOffset (MQLONG) 64
PutMsgRecFields (MQLONG) 64
PutMsgRecOffset (MQLONG) 65
RecsPresent (MQLONG) 65
StrucId (MQCHAR4) 65
StrucLength (MQLONG) 65
Version (MQLONG) 66
Initial values and language declarations 66
C declaration 66
COBOL declaration 67
PL/I declaration 67
Visual Basic declaration 67

Chapter 6. MQDLH - Dead-letter header 69
Overview 69
Fields 71
CodedCharSetId (MQLONG) 71
DestQMgrName (MQCHAR48) 71
DestQName (MQCHAR48) 72
Encoding (MQLONG) 72
Format (MQCHAR8) 72
PutApplName (MQCHAR28) 72
PutApplType (MQLONG) 73
PutDate (MQCHAR8) 73
PutTime (MQCHAR8) 73
Reason (MQLONG) 74
StrucId (MQCHAR4) 75
Version (MQLONG) 75
Initial values and language declarations 76
C declaration 76
COBOL declaration 77
PL/I declaration 77
System/390 assembler declaration 78
TAL declaration 78
Visual Basic declaration 78

Chapter 7. MQGMO - Get-message options 81
Overview 81
Fields 82
GroupStatus (MQCHAR) 82
MatchOptions (MQLONG) 82
MsgToken (MQBYTE16) 85
Options (MQLONG) 86
Reserved1 (MQCHAR) 108
ResolvedQName (MQCHAR48) 108
ReturnedLength (MQLONG) 109
Segmentation (MQCHAR) 109
SegmentStatus (MQCHAR) 109
Signal1 (MQLONG) 110
Signal2 (MQLONG) 111
StrucId (MQCHAR4) 111
Version (MQLONG) 111
WaitInterval (MQLONG) 112
Initial values and language declarations 113
C declaration 113
COBOL declaration 114
PL/I declaration 114
System/390 assembler declaration 115
TAL declaration 115
Visual Basic declaration 115

Chapter 8. MQIIH - IMS information header 117
Overview 117
Fields 118
Authenticator (MQCHAR8) 118
CodedCharSetId (MQLONG) 118
CommitMode (MQCHAR) 118
Encoding (MQLONG) 119
Flags (MQLONG) 119
Format (MQCHAR8) 119
LTermOverride (MQCHAR8) 119

MFSMapName (MQCHAR8) 119
ReplyToFormat (MQCHAR8) 119
Reserved (MQCHAR) 120
SecurityScope (MQCHAR) 120
StrucId (MQCHAR4) 120
StrucLength (MQLONG) 120
TranInstanceId (MQBYTE16) 121
TranState (MQCHAR) 121
Version (MQLONG) 121
Initial values and language declarations 122
C declaration 122
COBOL declaration 123
PL/I declaration 123
System/390 assembler declaration 123

Chapter 9. MQMD - Message descriptor 125
Overview. 125
Fields 127
AccountingToken (MQBYTE32) 127
ApplIdentityData (MQCHAR32) 129
ApplOriginData (MQCHAR4) 130
BackoutCount (MQLONG) 130
CodedCharSetId (MQLONG) 131
CorrelId (MQBYTE24) 132
Encoding (MQLONG) 133
Expiry (MQLONG) 134
Feedback (MQLONG) 136
Format (MQCHAR8) 140
GroupId (MQBYTE24) 146
MsgFlags (MQLONG) 147
MsgId (MQBYTE24) 151
MsgSeqNumber (MQLONG) 153
MsgType (MQLONG) 154
Offset (MQLONG). 155
OriginalLength (MQLONG) 156
Persistence (MQLONG) 156
Priority (MQLONG) 158
PutApplName (MQCHAR28) 159
PutApplType (MQLONG) 160
PutDate (MQCHAR8) 162
PutTime (MQCHAR8) 163
ReplyToQ (MQCHAR48) 164
ReplyToQMgr (MQCHAR48) 165
Report (MQLONG) 165
StrucId (MQCHAR4) 176
UserIdentifier (MQCHAR12) 176
Version (MQLONG) 178
Initial values and language declarations 178
C declaration 179
COBOL declaration 180
PL/I declaration 181
System/390 assembler declaration 182
TAL declaration 182
Visual Basic declaration 183

Chapter 10. MQMDE - Message descriptor extension 185
Overview. 185
Fields 187
CodedCharSetId (MQLONG) 188
Encoding (MQLONG) 188
Flags (MQLONG) 188
Format (MQCHAR8) 188
GroupId (MQBYTE24) 189
MsgFlags (MQLONG) 189
MsgSeqNumber (MQLONG) 189
Offset (MQLONG). 189
OriginalLength (MQLONG) 189
StrucId (MQCHAR4) 189
StrucLength (MQLONG) 189
Version (MQLONG) 190
Initial values and language declarations 190
C declaration 190
COBOL declaration 191
PL/I declaration 191
System/390 assembler declaration 192
Visual Basic declaration 192

Chapter 11. MQOD - Object descriptor 193
Overview. 193
Fields 194
AlternateSecurityId (MQBYTE40). 194
AlternateUserId (MQCHAR12) 195
DynamicQName (MQCHAR48) 196
InvalidDestCount (MQLONG). 196
KnownDestCount (MQLONG) 196
ObjectName (MQCHAR48) 197
ObjectQMgrName (MQCHAR48) 198
ObjectRecOffset (MQLONG) 199
ObjectRecPtr (MQPTR) 199
ObjectType (MQLONG) 200
RecsPresent (MQLONG). 200
ResolvedQMgrName (MQCHAR48) 201
ResolvedQName (MQCHAR48) 201
ResponseRecOffset (MQLONG) 201
ResponseRecPtr (MQPTR) 202
StrucId (MQCHAR4) 203
UnknownDestCount (MQLONG). 203
Version (MQLONG) 203
Initial values and language declarations 204
C declaration 204
COBOL declaration 205
PL/I declaration 205
System/390 assembler declaration 206
TAL declaration 206
Visual Basic declaration 206

Chapter 12. MQOR - Object record 209
Overview. 209
Fields 209
ObjectName (MQCHAR48) 209
ObjectQMgrName (MQCHAR48) 209
Initial values and language declarations 210
C declaration 210
COBOL declaration 210
PL/I declaration 210
Visual Basic declaration 210

Chapter 13. MQPMO - Put message options 211
Overview 211
Fields 212
Context (MQHOBJ) 212
InvalidDestCount (MQLONG) 212
KnownDestCount (MQLONG) 212
Options (MQLONG) 213
PutMsgRecFields (MQLONG) 222
PutMsgRecOffset (MQLONG) 223
PutMsgRecPtr (MQPTR) 224
RecsPresent (MQLONG) 224
ResolvedQMgrName (MQCHAR48) 225
ResolvedQName (MQCHAR48) 225
ResponseRecOffset (MQLONG) 225
ResponseRecPtr (MQPTR) 226
StrucId (MQCHAR4) 227
Timeout (MQLONG) 227
UnknownDestCount (MQLONG) 227
Version (MQLONG) 227
Initial values and language declarations 228
C declaration 228
COBOL declaration 229
PL/I declaration 229
System/390 assembler declaration 230
TAL declaration 230
Visual Basic declaration 231

Chapter 14. MQPMR - Put-message record 233
Overview 233
Fields 233
AccountingToken (MQBYTE32) 234
CorrelId (MQBYTE24) 234
Feedback (MQLONG) 234
GroupId (MQBYTE24) 234
MsgId (MQBYTE24) 235
Initial values and language declarations 235
C declaration 235
COBOL declaration 235
PL/I declaration 236
Visual Basic declaration 236

Chapter 15. MQRFH - Rules and formatting header 237
Overview 237
Fields 237
CodedCharSetId (MQLONG) 237
Encoding (MQLONG) 238
Flags (MQLONG) 238
Format (MQCHAR8) 238
NameValueString (MQCHARn) 238
StrucId (MQCHAR4) 239
StrucLength (MQLONG) 239
Version (MQLONG) 240
Initial values and language declarations 240
C declaration 240
COBOL declaration 241
PL/I declaration 241
System/390 assembler declaration 241

Chapter 16. MQRFH2 - Rules and formatting header version 2 243
Overview 243
Fields 244
CodedCharSetId (MQLONG) 244
Encoding (MQLONG) 244
Flags (MQLONG) 244
Format (MQCHAR8) 244
NameValueCCSID (MQLONG) 245
NameValueData (MQCHARn) 245
NameValueLength (MQLONG) 247
StrucId (MQCHAR4) 248
StrucLength (MQLONG) 248
Version (MQLONG) 248
Initial values and language declarations 249
C declaration 249
COBOL declaration 249
PL/I declaration 250
System/390 assembler declaration 250

Chapter 17. MQRMH - Reference message header 251
Overview 251
Fields 252
CodedCharSetId (MQLONG) 252
DataLogicalLength (MQLONG) 253
DataLogicalOffset (MQLONG) 253
DataLogicalOffset2 (MQLONG) 254
DestEnvLength (MQLONG) 254
DestEnvOffset (MQLONG) 254
DestNameLength (MQLONG) 254
DestNameOffset (MQLONG) 254
Encoding (MQLONG) 255
Flags (MQLONG) 255
Format (MQCHAR8) 255
ObjectInstanceId (MQBYTE24) 256
ObjectType (MQCHAR8) 256
SrcEnvLength (MQLONG) 256
SrcEnvOffset (MQLONG) 256
SrcNameLength (MQLONG) 257
SrcNameOffset (MQLONG) 257
StrucId (MQCHAR4) 257
StrucLength (MQLONG) 257
Version (MQLONG) 258
Initial values and language declarations 258
C declaration 259
COBOL declaration 259
PL/I declaration 260
System/390 assembler declaration 260
Visual Basic declaration 261

Chapter 18. MQRR - Response record 263
Overview 263
Fields 263
CompCode (MQLONG) 263
Reason (MQLONG) 263
Initial values and language declarations 264
C declaration 264
COBOL declaration 264
PL/I declaration 264

Visual Basic declaration 264

Chapter 19. MQTM - Trigger message 265
Overview. 265
Fields 267
ApplId (MQCHAR256) 267
ApplType (MQLONG) 267
EnvData (MQCHAR128). 268
ProcessName (MQCHAR48) 268
QName (MQCHAR48) 268
StrucId (MQCHAR4) 269
TriggerData (MQCHAR64) 269
UserData (MQCHAR128) 269
Version (MQLONG) 270
Initial values and language declarations 270
C declaration 270
COBOL declaration 271
PL/I declaration 271
System/390 assembler declaration 271
TAL declaration 271
Visual Basic declaration 272

Chapter 20. MQTMC2 - Trigger message 2 (character format) 273
Overview. 273
Fields 274
ApplId (MQCHAR256) 274
ApplType (MQCHAR4) 274
EnvData (MQCHAR128). 274
ProcessName (MQCHAR48) 274
QMgrName (MQCHAR48) 274
QName (MQCHAR48) 274
StrucId (MQCHAR4) 274
TriggerData (MQCHAR64) 274
UserData (MQCHAR128) 275
Version (MQCHAR4) 275
Initial values and language declarations 275
C declaration 275
COBOL declaration 276
PL/I declaration 276
System/390 assembler declaration 276
TAL declaration 277
Visual Basic declaration 277

Chapter 21. MQWIH - Work information header 279
Overview. 279
Fields 279
CodedCharSetId (MQLONG) 279
Encoding (MQLONG) 280
Flags (MQLONG) 280
Format (MQCHAR8) 280
MsgToken (MQBYTE16) 281
Reserved (MQCHAR32) 281
ServiceName (MQCHAR32) 281
ServiceStep (MQCHAR8) 281
StrucId (MQCHAR4) 281
StrucLength (MQLONG) 281
Version (MQLONG) 282
Initial values and language declarations 282
C declaration 282
COBOL declaration 283
PL/I declaration 283
System/390 assembler declaration 283

Chapter 22. MQXP - Exit parameter block (OS/390 only). 285
Overview. 285
Fields 285
ExitCommand (MQLONG) 285
ExitId (MQLONG). 286
ExitParmCount (MQLONG) 286
ExitReason (MQLONG) 286
ExitResponse (MQLONG) 287
ExitUserArea (MQBYTE16) 287
Reserved (MQLONG) 288
StrucId (MQCHAR4) 288
Version (MQLONG) 288
Language declarations 288
C declaration 288
COBOL declaration 288
PL/I declaration 289
System/390 assembler declaration 289

Chapter 23. MQXQH - Transmission queue header 291
Overview. 291
Fields 294
MsgDesc (MQMD1) 294
RemoteQMgrName (MQCHAR48) 294
RemoteQName (MQCHAR48). 294
StrucId (MQCHAR4) 295
Version (MQLONG) 295
Initial values and language declarations 295
C declaration 296
COBOL declaration 296
PL/I declaration 297
System/390 assembler declaration 298
TAL declaration 298
Visual Basic declaration 298

Part 2. Function calls 301

Chapter 24. Call descriptions 305
Conventions used in the call descriptions 305
Using the calls in the C language. 307
Declaring the Buffer parameter 307

Chapter 25. MQBACK - Back out changes 309
Syntax. 309
Parameters 309
Hconn (MQHCONN) – input 309
CompCode (MQLONG) – output. 309
Reason (MQLONG) – output 309
Usage notes 310
Language invocations 312
C invocation. 312
COBOL invocation 312

PL/I invocation (AIX, OS/2, OS/390, Windows NT) . . . 313
System/390 assembler invocation (OS/390 only) 313
TAL invocation (Tandem NSK only) 313
Visual Basic invocation (Windows only) . . . 313

Chapter 26. MQBEGIN - Begin unit of work 315
Syntax. 315
Parameters 315
Hconn (MQHCONN) – input 315
BeginOptions (MQBO) – input/output 315
CompCode (MQLONG) – output. 315
Reason (MQLONG) – output 315
Usage notes 316
Language invocations 317
C invocation. 318
COBOL invocation 318
PL/I invocation (AIX, OS/2, Windows NT) . . 318
Visual Basic invocation (Windows only) . . . 318

Chapter 27. MQCLOSE - Close object 319
Syntax. 319
Parameters 319
Hconn (MQHCONN) – input 319
Hobj (MQHOBJ) – input/output 319
Options (MQLONG) – input 319
CompCode (MQLONG) – output. 321
Reason (MQLONG) – output 321
Usage notes 322
Language invocations 324
C invocation. 324
COBOL invocation 324
PL/I invocation (AIX, OS/2, OS/390, VSE/ESA, Windows NT 325
System/390 assembler invocation (OS/390 only) 325
TAL invocation (Tandem NSK only) 325
Visual Basic invocation (Windows only) . . . 325

Chapter 28. MQCMIT - Commit changes 327
Syntax. 327
Parameters 327
Hconn (MQHCONN) – input 327
CompCode (MQLONG) – output. 327
Reason (MQLONG) – output 327
Usage notes 328
Language invocations 330
C invocation. 330
COBOL invocation 330
PL/I invocation (AIX, OS/2, OS/390, Windows NT) 330
System/390 assembler invocation (OS/390 only) 331
TAL invocation (Tandem NSK only) 331
Visual Basic invocation (Windows only) . . . 331

Chapter 29. MQCONN - Connect queue manager 333
Syntax. 333
Parameters 333
QMgrName (MQCHAR48) – input 333
Hconn (MQHCONN) – output 335
CompCode (MQLONG) – output. 336
Reason (MQLONG) – output 336
Usage notes 338
Language invocations 339
C invocation. 339
COBOL invocation 340
PL/I invocation (AIX, OS/2, OS/390, VSE/ESA, Windows NT) 340
System/390 assembler invocation (OS/390 only) 340
TAL invocation (Tandem NSK only) 340
Visual Basic invocation (Windows only) . . . 340

Chapter 30. MQCONNX - Connect queue manager (extended) 341
Syntax. 341
Parameters 341
QMgrName (MQCHAR48) – input 341
ConnectOpts (MQCNO) – input/output . . . 341
Hconn (MQHCONN) – output 341
CompCode (MQLONG) – output. 341
Reason (MQLONG) – output 341
Language invocations 342
C invocation. 342
COBOL invocation 342
PL/I invocation (AIX, OS/2, OS/390, Windows NT) 342
System/390 assembler invocation (OS/390 only) 343
Visual Basic invocation (Windows only) . . . 343

Chapter 31. MQDISC - Disconnect queue manager 345
Syntax. 345
Parameters 345
Hconn (MQHCONN) – input/output 345
CompCode (MQLONG) – output. 345
Reason (MQLONG) – output 346
Usage notes 347
Language invocations 348
C invocation. 348
COBOL invocation 348
PL/I invocation (AIX, OS/2, OS/390, VSE/ESA, Windows NT) 348
System/390 assembler invocation (OS/390 only) 348
TAL invocation (Tandem NSK only) 348
Visual Basic invocation (Windows only) . . . 348

Chapter 32. MQGET - Get message 349
Syntax. 349
Parameters 349
Hconn (MQHCONN) – input 349
Hobj (MQHOBJ) – input. 349
MsgDesc (MQMD) – input/output 349
GetMsgOpts (MQGMO) – input/output . . . 350
BufferLength (MQLONG) – input 350
Buffer (MQBYTE×BufferLength) – output . . . 350
DataLength (MQLONG) – output 351
CompCode (MQLONG) – output. 351

Reason (MQLONG) – output 351
Usage notes 355
Language invocations 359
C invocation. 359
COBOL invocation 359
PL/I invocation (AIX, OS/2, OS/390, VSE/ESA, Windows NT) 360
System/390 assembler invocation (OS/390 only) 360
TAL invocation (Tandem NSK only) 360
Visual Basic invocation (Windows only) . . . 360

Chapter 33. MQINQ - Inquire about object attributes 363
Syntax. 363
Parameters 363
Hconn (MQHCONN) – input 363
Hobj (MQHOBJ) – input. 363
SelectorCount (MQLONG) – input 363
Selectors (MQLONG×SelectorCount) – input 364
IntAttrCount (MQLONG) – input 367
IntAttrs (MQLONG×IntAttrCount) – output . . 367
CharAttrLength (MQLONG) – input 368
CharAttrs (MQCHAR×CharAttrLength) – output. 368
CompCode (MQLONG) – output. 368
Reason (MQLONG) – output 369
Usage notes 370
Language invocations 371
C invocation. 371
COBOL invocation 371
PL/I invocation (AIX, OS/2, OS/390, VSE/ESA, Windows NT) 372
System/390 assembler invocation (OS/390 only) 372
TAL invocation (Tandem NSK only) 372
Visual Basic invocation (Windows only) . . . 373

Chapter 34. MQOPEN - Open object 375
Syntax. 375
Parameters 375
Hconn (MQHCONN) – input 375
ObjDesc (MQOD) – input/output 375
Options (MQLONG) – input 376
Hobj (MQHOBJ) – output 382
CompCode (MQLONG) – output. 382
Reason (MQLONG) – output 383
Usage notes 385
Language invocations 391
C invocation. 391
COBOL invocation 391
PL/I invocation (AIX, OS/2, OS/390, VSE/ESA, Windows NT) 391
System/390 assembler invocation (OS/390 only) 392
TAL invocation (Tandem NSK only) 392
Visual Basic invocation (Windows only) . . . 392

Chapter 35. MQPUT - Put message 393
Syntax. 393
Parameters 393
Hconn (MQHCONN) – input 393
Hobj (MQHOBJ) – input. 393
MsgDesc (MQMD) – input/output 393
PutMsgOpts (MQPMO) – input/output . . . 394
BufferLength (MQLONG) – input 394
Buffer (MQBYTE×BufferLength) – input . . . 395
CompCode (MQLONG) – output. 395
Reason (MQLONG) – output 395
Usage notes 399
Language invocations 403
C invocation. 403
COBOL invocation 403
PL/I invocation (AIX, OS/2, OS/390, VSE/ESA, Windows NT) 404
System/390 assembler invocation (OS/390 only) 404
TAL invocation (Tandem NSK only) 404
Visual Basic invocation (Windows only) . . . 404

Chapter 36. MQPUT1 - Put one message 405
Syntax. 405
Parameters 405
Hconn (MQHCONN) – input 405
ObjDesc (MQOD) – input/output 405
MsgDesc (MQMD) – input/output 405
PutMsgOpts (MQPMO) – input/output . . . 406
BufferLength (MQLONG) – input 406
Buffer (MQBYTE×BufferLength) – input . . . 406
CompCode (MQLONG) – output. 406
Reason (MQLONG) – output 406
Usage notes 411
Language invocations 412
C invocation. 412
COBOL invocation 412
PL/I invocation (AIX, OS/2, OS/390, VSE/ESA, Windows NT) 413
System/390 assembler invocation (OS/390 only) 413
TAL invocation (Tandem NSK only) 413
Visual Basic invocation (Windows only) . . . 414

Chapter 37. MQSET - Set object attributes 415
Syntax. 415
Parameters 415
Hconn (MQHCONN) – input 415
Hobj (MQHOBJ) – input. 415
SelectorCount (MQLONG) – input 415
Selectors (MQLONG×SelectorCount) – input 415
IntAttrCount (MQLONG) – input 416
IntAttrs (MQLONG×IntAttrCount) – input . . 416
CharAttrLength (MQLONG) – input 417
CharAttrs (MQCHAR×CharAttrLength) – input 417
CompCode (MQLONG) – output. 417
Reason (MQLONG) – output 417
Usage notes 419
Language invocations 419
C invocation. 420
COBOL invocation 420
PL/I invocation (AIX, OS/2, OS/390, VSE/ESA, Windows NT) 420
System/390 assembler invocation (OS/390 only) 421
TAL invocation (Tandem NSK only) 421

Visual Basic invocation (Windows only) . . . 421

Chapter 38. MQSYNC - Synchronize statistics updates (Tandem NSK only). 423
Syntax. 423
Parameters 423
Language invocations 424
C language invocation 424
COBOL language invocation 424
TAL language invocation 424

Part 3. Attributes of objects 425

Chapter 39. Attributes for queues. . . 427
Overview. 428
AlterationDate (MQCHAR12) 430
AlterationTime (MQCHAR8) 430
BackoutRequeueQName (MQCHAR48). . . . 431
BackoutThreshold (MQLONG) 431
BaseQName (MQCHAR48). 431
CFStrucName (MQCHAR12) 432
ClusterName (MQCHAR48) 432
ClusterNamelist (MQCHAR48) 433
CreationDate (MQCHAR12) 433
CreationTime (MQCHAR8). 433
CurrentQDepth (MQLONG) 434
DefBind (MQLONG) 434
DefinitionType (MQLONG). 435
DefInputOpenOption (MQLONG) 436
DefPersistence (MQLONG). 436
DefPriority (MQLONG) 437
DistLists (MQLONG). 438
HardenGetBackout (MQLONG) 439
IndexType (MQLONG) 440
InhibitGet (MQLONG) 441
InhibitPut (MQLONG) 441
InitiationQName (MQCHAR48) 442
MaxMsgLength (MQLONG) 442
MaxQDepth (MQLONG) 443
MsgDeliverySequence (MQLONG) 443
OpenInputCount (MQLONG) 444
OpenOutputCount (MQLONG) 445
ProcessName (MQCHAR48) 445
QDepthHighEvent (MQLONG) 446
QDepthHighLimit (MQLONG) 446
QDepthLowEvent (MQLONG) 446
QDepthLowLimit (MQLONG). 447
QDepthMaxEvent (MQLONG) 447
QDesc (MQCHAR64). 448
QName (MQCHAR48) 448
QServiceInterval (MQLONG) 449
QServiceIntervalEvent (MQLONG) 449
QSGDisp (MQLONG) 450
QType (MQLONG) 450
RemoteQMgrName (MQCHAR48) 451
RemoteQName (MQCHAR48). 451
RetentionInterval (MQLONG) 452
Scope (MQLONG). 452
Shareability (MQLONG). 453
StorageClass (MQCHAR8) 454
TriggerControl (MQLONG). 454
TriggerData (MQCHAR64) 454
TriggerDepth (MQLONG) 455
TriggerMsgPriority (MQLONG) 455
TriggerType (MQLONG). 456
Usage (MQLONG) 456
XmitQName (MQCHAR48). 457

Chapter 40. Attributes for namelists 459
Overview. 459
AlterationDate (MQCHAR12) 459
AlterationTime (MQCHAR8) 459
NameCount (MQLONG) 460
NamelistDesc (MQCHAR64) 460
NamelistName (MQCHAR48) 460
Names (MQCHAR48×NameCount) 460
QSGDisp (MQLONG) 461

Chapter 41. Attributes for process definitions. 463
Overview. 463
AlterationDate (MQCHAR12) 463
AlterationTime (MQCHAR8) 463
ApplId (MQCHAR256) 464
ApplType (MQLONG) 464
EnvData (MQCHAR128). 465
ProcessDesc (MQCHAR64) 465
ProcessName (MQCHAR48) 466
QSGDisp (MQLONG) 466
UserData (MQCHAR128) 467

Chapter 42. Attributes for the queue manager 469
Overview. 469
AlterationDate (MQCHAR12) 470
AlterationTime (MQCHAR8) 470
AuthorityEvent (MQLONG) 471
ChannelAutoDef (MQLONG) 471
ChannelAutoDefEvent (MQLONG) 471
ChannelAutoDefExit (MQCHARn) 472
ClusterWorkloadData (MQCHAR32). 472
ClusterWorkloadExit (MQCHARn) 472
ClusterWorkloadLength (MQLONG) 473
CodedCharSetId (MQLONG) 473
CommandInputQName (MQCHAR48) 474
CommandLevel (MQLONG) 474
DeadLetterQName (MQCHAR48) 476
DefXmitQName (MQCHAR48) 477
DistLists (MQLONG). 477
IGQPutAuthority (MQLONG). 477
IGQUserId (MQLONG) 478
InhibitEvent (MQLONG) 479
IntraGroupQueuing (MQLONG) 479
LocalEvent (MQLONG) 480
MaxHandles (MQLONG) 480
MaxMsgLength (MQLONG) 480
MaxPriority (MQLONG) 481
MaxUncommittedMsgs (MQLONG) 481
PerformanceEvent (MQLONG) 482
Platform (MQLONG). 482

QMgrDesc (MQCHAR64) 483
QMgrIdentifier (MQCHAR48) 483
QMgrName (MQCHAR48) 483
QSGName (MQCHAR4) 484
RemoteEvent (MQLONG) 484
RepositoryName (MQCHAR48) 484
RepositoryNamelist (MQCHAR48) 485
StartStopEvent (MQLONG) 485
SyncPoint (MQLONG) 485
TriggerInterval (MQLONG) 486

Part 4. Appendixes 487

Appendix A. Return codes 489
Completion codes 489
Reason codes 489

Appendix B. MQSeries constants . . . 547
List of constants 547
MQ_* (Lengths of character string and byte fields) 547
MQACT_* (Accounting token) 548
MQACTT_* (Accounting token type) 548
MQAT_* (Application type) 549
MQBND_* (Binding) 549
MQBO_* (Begin options) 549
MQBO_* (Begin options structure identifier) . . 550
MQBO_* (Begin options version) 550
MQCA_* (Character attribute selector) 550
MQCADSD_* (CICS header ADS descriptor) 551
MQCC_* (Completion code) 551
MQCCSI_* (Coded character set identifier) . . 551
MQCCT_* (CICS header conversational task) 551
MQCFAC_* (CICS header facility) 552
MQCFUNC_* (CICS header function name) . . 552
MQCGWI_* (CICS header get-wait interval) . . 552
MQCI_* (Correlation identifier) 552
MQCIH_* (CICS header flags) 553
MQCIH_* (CICS header length) 553
MQCIH_* (CICS header structure identifier) . . 553
MQCIH_* (CICS header version) 553
MQCLT_* (CICS header link type) 553
MQCMDL_* (Command level) 553
MQCNO_* (Connect options) 554
MQCNO_* (Connect options structure identifier) 554
MQCNO_* (Connect options version) 554
MQCO_* (Close options) 554
MQCODL_* (CICS header output data length) 555
MQCRC_* (CICS header return code) 555
MQCSC_* (CICS header transaction start code) 555
MQCT_* (Connection tag) 555
MQCTES_* (CICS header task end status) . . . 555
MQCUOWC_* (CICS header unit-of-work control) 556
MQDCC_* (Convert-characters masks and factors) 556
MQDCC_* (Convert-characters options) . . . 556
MQDH_* (Distribution header structure identifier) 556
MQDH_* (Distribution header version) 557
MQDHF_* (Distribution header flags) 557
MQDL_* (Distribution list support) 557
MQDLH_* (Dead-letter header structure identifier) 557
MQDLH_* (Dead-letter header version) . . . 557
MQDXP_* (Data-conversion-exit parameter structure identifier) 557
MQDXP_* (Data-conversion-exit parameter structure version) 558
MQEC_* (Signal event-control-block completion code) 558
MQEI_* (Expiry interval) 558
MQENC_* (Encoding) 558
MQENC_* (Encoding masks) 558
MQENC_* (Encoding for packed-decimal integers) 559
MQENC_* (Encoding for floating-point numbers) 559
MQENC_* (Encoding for binary integers) . . . 559
MQEVR_* (Event reporting) 559
MQFB_* (Feedback) 559
MQFMT_* (Format) 560
MQGI_* (Group identifier) 561
MQGMO_* (Get message options) 561
MQGMO_* (Get message options structure identifier) 562
MQGMO_* (Get message options version) . . . 562
MQGS_* (Group status) 562
MQHC_* (Connection handle) 562
MQHO_* (Object handle) 562
MQIA_* (Integer attribute selector) 563
MQIAUT_* (IMS authenticator) 564
MQIAV_* (Integer attribute value) 564
MQICM_* (IMS commit mode) 564
MQIGQ_* (Intra-group queuing) 564
MQIGQPA_* (Intra-group queuing put authority) 564
MQIIH_* (IMS header flags) 565
MQIIH_* (IMS header length) 565
MQIIH_* (IMS header structure identifier) . . . 565
MQIIH_* (IMS header version) 565
MQISS_* (IMS security scope) 565
MQIT_* (Index type) 565
MQITII_* (IMS transaction instance identifier) 566
MQITS_* (IMS transaction state) 566
MQMD_* (Message descriptor structure identifier) 566
MQMD_* (Message descriptor version) 566
MQMDE_* (Message descriptor extension length) 566
MQMDE_* (Message descriptor extension structure identifier) 567
MQMDE_* (Message descriptor extension version) 567
MQMDEF_* (Message descriptor extension flags) 567
MQMDS_* (Message delivery sequence) . . . 567
MQMF_* (Message flags) 567
MQMF_* (Message-flags masks) 567
MQMI_* (Message identifier) 568
MQMO_* (Match options) 568

MQMT_* (Message type) 568
MQMTOK_* (Message token) 568
MQNC_* (Name count) 569
MQOD_* (Object descriptor length) 569
MQOD_* (Object descriptor structure identifier) 569
MQOD_* (Object descriptor version) 569
MQOII_* (Object instance identifier) 569
MQOL_* (Original length) 569
MQOO_* (Open options) 570
MQOT_* (Object type) 570
MQPER_* (Persistence) 570
MQPL_* (Platform) 570
MQPMO_* (Put message options) 571
MQPMO_* (Put message options structure length) 571
MQPMO_* (Put message options structure identifier). 571
MQPMO_* (Put message options version) . . . 571
MQPMRF_* (Put message record field flags) . . 571
MQPRI_* (Priority) 572
MQQA_* (Inhibit get) 572
MQQA_* (Inhibit put) 572
MQQA_* (Backout hardening). 572
MQQA_* (Queue shareability). 572
MQQDT_* (Queue definition type) 572
MQQSGD_* (Queue-sharing group disposition) 573
MQQSIE_* (Service interval events) 573
MQQT_* (Queue type) 573
MQRC_* (Reason code) 573
MQRFH_* (Rules and formatting header flags) 579
MQRFH_* (Rules and formatting header length) 579
MQRFH_* (Rules and formatting header structure identifier) 579
MQRFH_* (Rules and formatting header version) 579
MQRL_* (Returned length) 579
MQRMH_* (Reference message header structure identifier). 579
MQRMH_* (Reference message header version) 580
MQRMHF_* (Reference message header flags) 580
MQRO_* (Report options) 580
MQRO_* (Report-options masks) 580
MQSCO_* (Queue scope) 580
MQSEG_* (Segmentation) 581
MQSID_* (Security identifier) 581
MQSIDT_* (Security identifier type) 581
MQSP_* (Syncpoint) 581
MQSS_* (Segment status) 581
MQTC_* (Trigger control) 581
MQTM_* (Trigger message structure identifier) 582
MQTM_* (Trigger message structure version) 582
MQTMC_* (Trigger message character format structure identifier) 582
MQTMC_* (Trigger message character format structure version) 582
MQTT_* (Trigger type) 582
MQUS_* (Usage) 583
MQWI_* (Wait interval) 583
MQWIH_* (Workload information header flags) 583
MQWIH_* (Workload information header structure length) 583
MQWIH_* (Workload information header structure identifier) 583
MQWIH_* (Workload information header version) 583
MQXC_* (Exit command identifier) 584
MQXCC_* (Exit response) 584
MQXDR_* (Data-conversion-exit response) . . 584
MQXP_* (Exit parameter block structure identifier). 584
MQXP_* (Exit parameter block version) . . . 584
MQXQH_* (Transmission queue header structure identifier) 585
MQXQH_* (Transmission queue header version) 585
MQXR_* (Exit reason) 585
MQXT_* (Exit identifier). 585
MQXUA_* (Exit user area) 585

Appendix C. Rules for validating MQI options 587
MQOPEN call 587
MQPUT call 587
MQPUT1 call 588
MQGET call 588
MQCLOSE call 588

Appendix D. Machine encodings . . . 589
Binary-integer encoding 589
Packed-decimal-integer encoding 590
Floating-point encoding 590
Constructing encodings 591
Analyzing encodings 591
Using bit operations 591
Using arithmetic 592
Summary of machine architecture encodings . . . 592

Appendix E. Report options and message flags 593
Structure of the report field. 593
Analyzing the report field 594
Using bit operations 595
Using arithmetic 595
Structure of the message-flags field 596

Appendix F. Data conversion. 599
Conversion processing 599
Processing conventions 601
Conversion of report messages 605
MQDXP – Data-conversion exit parameter. . . . 607
Overview. 607
Fields 608
AppOptions (MQLONG) 608
CodedCharSetId (MQLONG) 608
CompCode (MQLONG) 608
DataLength (MQLONG). 608
Encoding (MQLONG) 609
ExitOptions (MQLONG). 609
ExitResponse (MQLONG) 609
Hconn (MQHCONN). 610
Reason (MQLONG) 610
StrucId (MQCHAR4) 612

Version (MQLONG) 612
C declaration 612
COBOL declaration (AS/400 only) 613
System/390 assembler declaration (OS/390 only) 613
MQXCNVC - Convert characters 613
Syntax. 614
Parameters 614
C invocation. 619
COBOL invocation (AS/400 only) 619
System/390 assembler invocation (OS/390 only) 619
MQ_DATA_CONV_EXIT - Data conversion exit 620
Syntax. 620
Parameters 620
Usage notes 621
C invocation. 624
COBOL invocation (AS/400 only) 624
System/390 assembler invocation (OS/390 only) 624

Appendix G. Signal notification IPC message (Tandem NSK only) 627

Appendix H. Code page conversion tables. 629
Codeset names and CCSIDs 630
Code page conversion tables 631
OS/390 conversion support 666
OS/2 conversion support 682
OS/400 conversion support 683
Unicode conversion support 683
MQSeries OS/2 support for Unicode 683
MQSeries AIX support for Unicode 683
MQSeries HP-UX support for Unicode 684
MQSeries NT, Solaris and Tru64 support for Unicode 684
OS/400 support for Unicode 685
MQSeries for OS/390 support for Unicode . . 685

Appendix I. Notices. 687
Trademarks 688

Glossary of terms and abbreviations 691

Bibliography. 703
MQSeries cross-platform publications 703
MQSeries platform-specific publications 703
Softcopy books 704
HTML format 704
Portable Document Format (PDF) 704
BookManager® format 705
PostScript format 705
Windows Help format 705
MQSeries information available on the Internet . . 705

Index 707

Sending your comments to IBM . . . 715

Tables

1. Short names used for supported environments xviii
2. C and C++ language compilers xx
3. COBOL language compilers xxi
4. PL/I language compilers xxi
5. Visual Basic language compilers xxi
6. Assembler/390 language compilers xxii
7. TAL compilers xxii
8. Elementary data types in C 9
9. Elementary data types in COBOL 10
10. Elementary data types in PL/I 10
11. Elementary data types in System/390 assembler 11
12. Elementary data types in TAL 12
13. Elementary data types in Visual Basic. . . . 12
14. Structure data types used on MQI calls 14
15. Structure data types used in message data 14
16. C header files 16
17. COBOL COPY files 19
18. PL/I INCLUDE files 22
19. Assembler macros 23
20. Visual Basic header files 26
21. Fields in MQBO 29
22. Initial values of fields in MQBO. 30
23. Fields in MQCIH. 33
24. Contents of error information fields in MQCIH structure 35
25. Initial values of fields in MQCIH 46
26. Fields in MQCNO 51
27. Initial values of fields in MQCNO 58
28. Fields in MQDH 61
29. Initial values of fields in MQDH 66
30. Fields in MQDLH 69
31. Initial values of fields in MQDLH 76
32. Fields in MQGMO 81
33. MQGET options relating to messages in groups and segments of logical messages . . 101
34. Outcome when MQGET or MQCLOSE call is not consistent with group and segment information 103
35. Initial values of fields in MQGMO 113
36. Fields in MQIIH 117
37. Initial values of fields in MQIIH 122
38. Fields in MQMD 125
39. Initial values of fields in MQMD 178
40. Fields in MQMDE 185
41. Queue-manager action when MQMDE specified on MQPUT or MQPUT1. 186
42. Initial values of fields in MQMDE 190
43. Fields in MQOD 193
44. Initial values of fields in MQOD 204
45. Fields in MQOR 209
46. Initial values of fields in MQOR 210
47. Fields in MQPMO 211
48. MQPUT options relating to messages in groups and segments of logical messages . . 217
49. Outcome when MQPUT or MQCLOSE call not consistent with group and segment information 219
50. Initial values of fields in MQPMO 228
51. Fields in MQPMR 233
52. Fields in MQRFH 237
53. Initial values of fields in MQRFH 240
54. Fields in MQRFH2. 243
55. Initial values of fields in MQRFH2 249
56. Fields in MQRMH 251
57. Initial values of fields in MQRMH 258
58. Fields in MQRR. 263
59. Initial values of fields in MQRR 264
60. Fields in MQTM 265
61. Initial values of fields in MQTM 270
62. Fields in MQTMC2 273
63. Initial values of fields in MQTMC2 275
64. Fields in MQWIH 279
65. Initial values of fields in MQWIH. 282
66. Fields in MQXP. 285
67. Fields in MQXQH 291
68. Initial values of fields in MQXQH 295
69. Effect of MQCLOSE options on various types of object and queue 321
70. MQINQ attribute selectors for queues 364
71. MQINQ attribute selectors for namelists 366
72. MQINQ attribute selectors for process definitions 366
73. MQINQ attribute selectors for the queue manager 366
74. Valid MQOPEN options for each queue type 381
75. MQSET attribute selectors for queues 416
76. Attributes for queues 428
77. Recommended values of queue index type 440
78. Attributes for namelists 459
79. Attributes for process definitions 463
80. Attributes for the queue manager. 469
81. Summary of encodings for machine architectures 592
82. Fields in MQDXP 607
83. Codeset names and CCSIDs. 630
84. Conversion support: US ENGLISH 632
85. Conversion support: GERMAN 634
86. Conversion support: DANISH and NORWEGIAN 635
87. Conversion support: FINNISH and SWEDISH 637
88. Conversion support: ITALIAN 638
89. Conversion support: SPANISH. 639
90. Conversion support: UK ENGLISH / GAELIC 640
91. Conversion support: FRENCH 641
92. Conversion support: MULTILINGUAL 642
93. Conversion support: PORTUGUESE 643
94. Conversion support: ICELANDIC. 644
95. Conversion support: EASTERN EUROPEAN Languages 645

© Copyright IBM Corp. 1994, 2000

96. Conversion support: CYRILLIC 646
97. Conversion support: ESTONIAN 647
98. Conversion support: LATVIAN and LITHUANIAN 648
99. Conversion support: UKRAINIAN 649
100. Conversion support: GREEK 650
101. Conversion support: TURKISH 651
102. Conversion support: HEBREW. 652
103. Conversion support: ARABIC 653
104. Conversion support: FARSI 654
105. Conversion support: URDU. 655
106. Conversion support: THAI 656
107. Conversion support: LAO 657
108. Conversion support: VIETNAMESE 658
109. Conversion support: JAPANESE LATIN SBCS 659
110. Conversion support: JAPANESE KATAKANA SBCS 660
111. Conversion support: JAPANESE KANJI / LATIN MIXED 661
112. Conversion support: JAPANESE KANJI / KATAKANA MIXED 662
113. Conversion support: KOREAN. 663
114. Conversion support: SIMPLIFIED CHINESE 664
115. Conversion support: TRADITIONAL CHINESE. 665
116. MQSeries for OS/390 CCSID conversion support 666

About this book

The IBM® MQSeries 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 MQSeries programming interface, the MQI, for the following products:

MQSeries for AIX, V5.1
MQSeries for AS/400, V5.1
MQSeries for AT&T GIS UNIX V2.2 [1]
MQSeries for Compaq Tru64 UNIX, V5.1
MQSeries for Compaq (DIGITAL) OpenVMS, V2.2.1.1
MQSeries for HP-UX, V5.1
MQSeries for OS/2 Warp, V5.1
MQSeries for OS/390, V2.2
MQSeries for SINIX and DC/OSx, V2.2
MQSeries for Sun Solaris, V5.1 (SPARC and Intel Platform Editions)
MQSeries for Tandem NonStop Kernel, V2.2.0.1
MQSeries for VSE/ESA V2.1
MQSeries for Windows NT, V5.1
MQSeries for Windows V2.0
MQSeries for Windows V2.1

Notes:

1. This book does not apply to the MQSeries for AS/400® Version 5 Release 1 product using the RPG programming language.

   You should use the *MQSeries for AS/400 Version 5 Release 1 Application Programming Reference (ILE RPG)* manual, SC34-5559 for this programming language.

2. C++

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

For information on how to design and write applications that use the services MQSeries provides, see the *MQSeries 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.

What you need to know to understand this book

To write message queuing applications using MQSeries, 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®, OS/390®, VSE/ESA™, and Windows NT)
- System/390® assembler (available on OS/390 only)

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

© Copyright IBM Corp. 1994, 2000

- TAL (available on Tandem NonStop Kernel only)
- Visual Basic V4 or V5 (available on Windows® 3.1, Windows 95, Windows 98, and Windows 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:

MQSeries The MQSeries 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	MQSeries for AIX Version 5.1
DOS client	MQ client applications running on PC-DOS
HP-UX	MQSeries for HP-UX Version 5.1
OS/390	MQSeries for OS/390 Version 2.2
Compaq (DIGITAL) OpenVMS	MQSeries for Digital OpenVMS Version 2.2
OS/2	MQSeries for OS/2 Warp Version 5.1
AS/400	MQSeries for AS/400 Version 5R1
Sun Solaris	MQSeries for Sun Solaris Version 5.1
Tandem NonStop Kernel	MQSeries for Tandem NonStop Kernel Version 2.2
UNIX systems	The UNIX systems supported by MQSeries that are not Version 5. These are: • MQSeries for AT&T GIS UNIX, Version 2.2 • MQSeries for SINIX and DC/OSx Version 2.2 • MQSeries for DIGITAL UNIX (Compaq Tru64 UNIX)
Windows client	MQSeries client applications running on Windows 3.1, Windows 95, Windows 98, or Windows NT
Windows NT	MQSeries for Windows NT Version 5.1
Windows 3.1	MQSeries for Windows Version 2.0
Windows 95, Windows 98	MQSeries for Windows Version 2.1

The following table lists the MQSeries products available for Windows, and shows the Windows platforms on which each runs.

Product	Windows 3.1	Windows 95	Windows 98	Windows NT
MQSeries for Windows Client	Yes	Yes	Yes	Yes
MQSeries for Windows NT	No	No	No	Yes
MQSeries for Windows V2.0	Yes	Yes	No	No
MQSeries for Windows V2.1	No	Yes	Yes	Yes

MQSeries for Windows Versions 2.0 and 2.1 support most of the features of the MQI described in this book. For information on these products, see the *MQSeries for Windows User's Guide*.

Language compilers

Also, we use the following shortened names for these language compilers:

- C – see Table 2 on page xx
- COBOL – see Table 3 on page xxi
- PL/I – see Table 4 on page xxi
- Visual Basic – see Table 5 on page xxi
- Assembler/390 – see Table 6 on page xxii
- TAL – see Table 7 on page xxii

Table 2. C and C++ language compilers

Platform	Compiler
AIX	IBM C for AIX Version 3.1.4 IBM C Set++ for AIX V3.1
AIX C++	IBM C Set++ for AIX V3.1
AS/400	IBM ILE C for AS/400, V4R4M0
AS/400 C++	IBM VisualAge® C++ compiler for AS/400, 5716-CX4 PRPQ ILE C++ compiler for AS/400, 5799-GDW
AT&T	AT&T GIS High Performance C V1.0b compiler
AT&T C++	AT&T C++ language system for AT&T GIS UNIX
DC/OSx	DC/OSx C4.0 Version 4.0.1 compiler
Digital OpenVMS	DEC C Version 5.0
Digital OpenVMS C++	DEC C++ V5.0 (VAX) V5.2 (AXP)
Digital UNIX	DEC C V5.2 for Digital UNIX
HP-UX	C Softbench Version 5.0 HP-UX ANSI C HP C++, V3.1 for HP-UX V10.x HP C, V3.6 for HP-UX
HP-UX C++	HP C++, V3.1 for HP-UX V10.x IBM C and C++ compiler, V3.6 HP-UX ANSI C++ for V10 and V11 ANSI C++ compiler V3.6
OS/2	IBM VisualAge for C++ for OS/2 V3.0 Borland C++ V2.0 (C bindings only) IBM C and C++ compiler, V3.6
OS/2 C++	IBM VisualAge for C++ for OS/2, V3.0
OS/390	C/370™ Release 2.1.0 IBM SAA® AD/Cycle® C/370 Compiler
OS/390 C++	IBM OS/390 C/C++ V2R4 IBM OS/390 C/C++
SINIX	C compiler (C-DS, MIPS) V1.1
Sun Solaris	Sun WorkShop Compiler C V4.2
Sun Solaris C++	Sun WorkShop Compiler C++ V4.2
Tandem NSK	D30 or later using the WIDE memory model (32-bit integers)
VSE/ESA	IBM C for VSE/ESA V1.1
Windows NT	Microsoft Visual C++ for Windows 95 and Windows NT V4.0 IBM VisualAge for C++ for Windows V3.5
Windows NT C++	IBM VisualAge for C++ for Windows V3.5 Microsoft Visual C++ for Windows 95 and Windows NT V4.0 IBM VisualAge for C++ Professional V4.0 IBM VisualAge for C++ Professional V5.0 IBM C and C++ compiler, V3.6.4
MQSeries for Windows V2.0 - 16-bit	16-bit C - Microsoft® Visual C++ V1.5
MQSeries for Windows V2.0- 32-bit	32-bit C - Microsoft Visual C++ V2.0
MQSeries for Windows V2.1	Microsoft Visual C++ V4.0 Borland C
DOS clients	Microsoft C V7 Microsoft Visual C++ V1.5
Windows 3.1 clients	Microsoft C V7.0
Windows 3.1 clients C++	Microsoft Visual C++ V2.0
Windows 95 and Windows 98 clients	Microsoft Visual C++ V2.0

Table 2. C and C++ language compilers (continued)

Platform	Compiler
Windows 95 and Windows 98 clients C++	IBM VisualAge for C++ V3.5 Microsoft Visual C++ V4.0
Note: AT&T has become NCR UNIX SVR4 MP-RAS, R3.0	

Table 3. COBOL language compilers

Platform	Compiler
AIX	The Micro Focus COBOL compiler V4.0 for UNIX Systems IBM COBOL Set for AIX Version 1.1
AS/400	IBM ILE COBOL compiler for AS/400 V4R4M0
Digital OpenVMS	DEC COBOL V5.0 (VAX) V2.2 (AXP)
HP-UX	COBOL Softbench Version 4.0 Micro Focus COBOL compiler Version 4.0 for UNIX Systems
OS/2	Micro Focus COBOL compiler V4.0 IBM VisualAge for COBOL for OS/2 V1.1
OS/390	IBM COBOL for MVS™ and VM (formerly COBOL/370) IBM COBOL for OS/390 and VM
SINIX and DC/OSx	Micro Focus COBOL compiler V3.2 for SINIX
Sun Solaris	Micro Focus COBOL compiler for UNIX systems V4.0
Tandem NSK	D30 or later
VSE/ESA	IBM COBOL for VSE/ESA V1.1
Windows NT	Micro Focus Object COBOL compiler V3.3 or V4.0 for Windows NT IBM VisualAge COBOL Enterprise V2.2 IBM VisualAge COBOL for Windows NT V2.1
Windows 95 and Windows 98 clients	Micro Focus COBOL Workbench V4.0

Table 4. PL/I language compilers

Platform	Compiler
AIX	IBM PL/I Set for AIX V1.1
OS/2	IBM Visual Age for PL/I for OS/2 IBM PL/I for OS/2 V1.2
OS/390	IBM SAA AD/Cycle PL/I IBM PL/I for MVS and VM
VSE/ESA	IBM PL/I for VSE/ESA V1.1
Windows NT	IBM Visual Age for PL/I for Windows IBM PL/I for Windows V1.2 IBM VisualAge PL/I Enterprise V2.1

In addition, MQSeries for Windows V2.0, MQSeries for Windows V2.1, and MQSeries for Windows NT, V5.1 support Basic compilers.

Table 5. Visual Basic language compilers

Platform	Compiler
MQSeries for Windows V2.0 - 16-bit	Microsoft Visual Basic V4.0 (16 bit)
MQSeries for Windows V2.0 - 32-bit	Microsoft Visual Basic V4.0 (32 bit)

Table 5. Visual Basic language compilers (continued)

Platform	Compiler
MQSeries for Windows V2.1	Microsoft Visual Basic V4.0
MQSeries for Windows NT V5.1	Microsoft Visual Basic V4.0 or V5.0
Windows 3.1 clients	Microsoft Visual Basic V4.0
Windows 95 and Windows 98 clients	Microsoft Visual Basic V4.0 or V5.0

Table 6. Assembler/390 language compilers

Platform	Compiler
OS/390	Assembler H assembler IBM High Level Assembler/MVS assembler

Table 7. TAL compilers

Platform	Compiler
Tandem NSK	D30 or later IBM High Level Assembler/MVS assembler

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 MQSeries 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 MQSeries objects
- Values of constants you need to use when you write MQSeries programs
- Reason codes that may occur when you run your programs

There is a glossary and a bibliography at the back of the book.

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

Summary of changes

This section describes changes in this edition of *MQSeries 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 (SC33-1673-07)

Major changes for this edition include:

- 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.
- 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.

Changes for the previous edition included:

- Addition of the MQSeries for AS/400 V5R1 product.

Changes for the sixth edition included:

- Addition of the following versions and releases of MQSeries products:
  - MQSeries for AIX, V5.1
  - MQSeries for AS/400 V4R2M1
  - MQSeries for HP-UX, V5.1
  - MQSeries for OS/2 Warp, V5.1
  - MQSeries for OS/390, V2.1
  - MQSeries for Sun Solaris, V5.1
  - MQSeries for VSE/ESA V2.1
  - MQSeries for Windows NT, V5.1
- Addition of the MQWIH structure
- Addition of cluster support
- Addition of code pages supporting the euro currency symbol

© Copyright IBM Corp. 1994, 2000

Changes

Part 1. Data type descriptions

Chapter 1. Introduction 7
Elementary data types 7
MQBYTE – Byte 7
MQBYTEn – String of n bytes 7
MQCHAR – Single-byte character 8
MQCHARn – String of n single-byte characters . . 8
MQHCONN – Connection handle 8
MQHOBJ – Object handle 8
MQLONG – Long integer 9
MQPTR – Pointer 9
C declarations 9
COBOL declarations 10
PL/I declarations (AIX, OS/2, OS/390, VSE/ESA, and Windows NT only) 10
System/390 Assembler declarations (OS/390 only) 11
TAL declarations (Tandem NonStop Kernel only) 12
Visual Basic declarations (Windows 3.1, Windows 95, Windows 98, and Windows NT) 12
Structure data types – introduction 14
Summary 14
Rules for structure data types 14
Conventions used in the descriptions 15
C programming 16
Header files 16
Functions 16
Parameters with undefined data type 17
Data types 17
Manipulating binary strings 17
Manipulating character strings 17
Initial values for structures 18
Initial values for dynamic structures 18
Use from C++ 19
Notational conventions 19
COBOL programming 19
COPY files 19
Structures 20
Pointers 21
Named constants 21
Notational conventions 22
PL/I programming 22
INCLUDE files 22
Structures 22
Named constants 23
Notational conventions 23
System/390 Assembler programming 23
Macros 23
Structures 24
Specifying the name of the structure 24
Specifying the form of the structure 24
Controlling the version of the structure . . . 25
Declaring one structure embedded within another 25
Specifying initial values for fields 25
Controlling the listing 26
CMQVERA macro 26
Notational conventions 26
Visual Basic programming 26
Header files 26
Parameters of the MQI calls 27
Initial values for structures 27
Notational conventions 27

Chapter 2. MQBO - Begin options 29
Overview 29
Fields 29
Options (MQLONG) 29
StrucId (MQCHAR4) 29
Version (MQLONG) 30
Initial values and language declarations 30
C declaration 30
COBOL declaration 30
PL/I declaration 31
Visual Basic declaration 31

Chapter 3. MQCIH - CICS information header . . 33
Overview 34
Fields 35
AbendCode (MQCHAR4) 35
ADSDescriptor (MQLONG) 35
AttentionId (MQCHAR4) 36
Authenticator (MQCHAR8) 36
CancelCode (MQCHAR4) 36
CodedCharSetId (MQLONG) 37
CompCode (MQLONG) 37
ConversationalTask (MQLONG) 37
CursorPosition (MQLONG) 37
Encoding (MQLONG) 37
ErrorOffset (MQLONG) 37
Facility (MQBYTE8) 38
FacilityKeepTime (MQLONG) 38
FacilityLike (MQCHAR4) 38
Flags (MQLONG) 38
Format (MQCHAR8) 39
Function (MQCHAR4) 39
GetWaitInterval (MQLONG) 40
InputItem (MQLONG) 40
LinkType (MQLONG) 40
NextTransactionId (MQCHAR4) 40
OutputDataLength (MQLONG) 40
Reason (MQLONG) 41
RemoteSysId (MQCHAR4) 41
RemoteTransId (MQCHAR4) 41
ReplyToFormat (MQCHAR8) 41
Reserved1 (MQCHAR8) 42
Reserved2 (MQCHAR8) 42
Reserved3 (MQCHAR8) 42
Reserved4 (MQLONG) 42
ReturnCode (MQLONG) 42
StartCode (MQCHAR4) 43
StrucId (MQCHAR4) 43
StrucLength (MQLONG) 44

© Copyright IBM Corp. 1994, 2000

TaskEndStatus (MQLONG) 44
TransactionId (MQCHAR4) 44
UOWControl (MQLONG) 45
Version (MQLONG) 45
Initial values and language declarations 46
C declaration 47
COBOL declaration 48
PL/I declaration 49
System/390 assembler declaration 50

Chapter 4. MQCNO - Connect options 51
Overview 51
Fields 51
ClientConnOffset (MQLONG) 52
ClientConnPtr (MQPTR) 52
ConnTag (MQBYTE128) 54
Options (MQLONG) 54
StrucId (MQCHAR4) 57
Version (MQLONG) 57
Initial values and language declarations 58
C declaration 58
COBOL declaration 58
PL/I declaration 59
System/390 assembler declaration (OS/390) . . 59
Visual Basic declaration 59

Chapter 5. MQDH - Distribution header 61
Overview 61
Fields 62
CodedCharSetId (MQLONG) 62
Encoding (MQLONG) 63
Flags (MQLONG) 63
Format (MQCHAR8) 64
ObjectRecOffset (MQLONG) 64
PutMsgRecFields (MQLONG) 64
PutMsgRecOffset (MQLONG) 65
RecsPresent (MQLONG) 65
StrucId (MQCHAR4) 65
StrucLength (MQLONG) 65
Version (MQLONG) 66
Initial values and language declarations 66
C declaration 66
COBOL declaration 67
PL/I declaration 67
Visual Basic declaration 67

Chapter 6. MQDLH - Dead-letter header 69
Overview 69
Fields 71
CodedCharSetId (MQLONG) 71
DestQMgrName (MQCHAR48) 71
DestQName (MQCHAR48) 72
Encoding (MQLONG) 72
Format (MQCHAR8) 72
PutApplName (MQCHAR28) 72
PutApplType (MQLONG) 73
PutDate (MQCHAR8) 73
PutTime (MQCHAR8) 73
Reason (MQLONG) 74
StrucId (MQCHAR4) 75
Version (MQLONG) 75
Initial values and language declarations 76
C declaration 76
COBOL declaration 77
PL/I declaration 77
System/390 assembler declaration 78
TAL declaration 78
Visual Basic declaration 78

Chapter 7. MQGMO - Get-message options . . . 81
Overview 81
Fields 82
GroupStatus (MQCHAR) 82
MatchOptions (MQLONG) 82
MsgToken (MQBYTE16) 85
Options (MQLONG) 86
Reserved1 (MQCHAR) 108
ResolvedQName (MQCHAR48) 108
ReturnedLength (MQLONG) 109
Segmentation (MQCHAR) 109
SegmentStatus (MQCHAR) 109
Signal1 (MQLONG) 110
Signal2 (MQLONG) 111
StrucId (MQCHAR4) 111
Version (MQLONG) 111
WaitInterval (MQLONG) 112
Initial values and language declarations 113
C declaration 113
COBOL declaration 114
PL/I declaration 114
System/390 assembler declaration 115
TAL declaration 115
Visual Basic declaration 115

Chapter 8. MQIIH - IMS information header . . 117
Overview 117
Fields 118
Authenticator (MQCHAR8) 118
CodedCharSetId (MQLONG) 118
CommitMode (MQCHAR) 118
Encoding (MQLONG) 119
Flags (MQLONG) 119
Format (MQCHAR8) 119
LTermOverride (MQCHAR8) 119
MFSMapName (MQCHAR8) 119
ReplyToFormat (MQCHAR8) 119
Reserved (MQCHAR) 120
SecurityScope (MQCHAR) 120
StrucId (MQCHAR4) 120
StrucLength (MQLONG) 120
TranInstanceId (MQBYTE16) 121
TranState (MQCHAR) 121
Version (MQLONG) 121
Initial values and language declarations 122
C declaration 122
COBOL declaration 123
PL/I declaration 123
System/390 assembler declaration 123

Chapter 9. MQMD - Message descriptor 125
Overview 125
Fields 127

AccountingToken (MQBYTE32) 127
ApplIdentityData (MQCHAR32) 129
ApplOriginData (MQCHAR4) 130
BackoutCount (MQLONG) 130
CodedCharSetId (MQLONG) 131
CorrelId (MQBYTE24) 132
Encoding (MQLONG) 133
Expiry (MQLONG) 134
Feedback (MQLONG) 136
Format (MQCHAR8) 140
GroupId (MQBYTE24) 146
MsgFlags (MQLONG) 147
MsgId (MQBYTE24) 151
MsgSeqNumber (MQLONG) 153
MsgType (MQLONG) 154
Offset (MQLONG). 155
OriginalLength (MQLONG) 156
Persistence (MQLONG) 156
Priority (MQLONG) 158
PutApplName (MQCHAR28) 159
PutApplType (MQLONG) 160
PutDate (MQCHAR8) 162
PutTime (MQCHAR8) 163
ReplyToQ (MQCHAR48) 164
ReplyToQMgr (MQCHAR48) 165
Report (MQLONG) 165
StrucId (MQCHAR4) 176
UserIdentifier (MQCHAR12) 176
Version (MQLONG) 178
Initial values and language declarations 178
C declaration 179
COBOL declaration 180
PL/I declaration 181
System/390 assembler declaration 182
TAL declaration 182
Visual Basic declaration 183

Chapter 10. MQMDE - Message descriptor extension 185
Overview. 185
Fields 187
CodedCharSetId (MQLONG) 188
Encoding (MQLONG) 188
Flags (MQLONG) 188
Format (MQCHAR8) 188
GroupId (MQBYTE24) 189
MsgFlags (MQLONG) 189
MsgSeqNumber (MQLONG) 189
Offset (MQLONG). 189
OriginalLength (MQLONG) 189
StrucId (MQCHAR4) 189
StrucLength (MQLONG) 189
Version (MQLONG) 190
Initial values and language declarations 190
C declaration 190
COBOL declaration 191
PL/I declaration 191
System/390 assembler declaration 192
Visual Basic declaration 192

Chapter 11. MQOD - Object descriptor 193
Overview. 193
Fields 194
AlternateSecurityId (MQBYTE40). 194
AlternateUserId (MQCHAR12) 195
DynamicQName (MQCHAR48) 196
InvalidDestCount (MQLONG). 196
KnownDestCount (MQLONG) 196
ObjectName (MQCHAR48). 197
ObjectQMgrName (MQCHAR48). 198
ObjectRecOffset (MQLONG) 199
ObjectRecPtr (MQPTR) 199
ObjectType (MQLONG) 200
RecsPresent (MQLONG). 200
ResolvedQMgrName (MQCHAR48) 201
ResolvedQName (MQCHAR48) 201
ResponseRecOffset (MQLONG) 201
ResponseRecPtr (MQPTR) 202
StrucId (MQCHAR4) 203
UnknownDestCount (MQLONG). 203
Version (MQLONG) 203
Initial values and language declarations 204
C declaration 204
COBOL declaration 205
PL/I declaration 205
System/390 assembler declaration 206
TAL declaration 206
Visual Basic declaration 206

Chapter 12. MQOR - Object record 209
Overview. 209
Fields 209
ObjectName (MQCHAR48). 209
ObjectQMgrName (MQCHAR48). 209
Initial values and language declarations 210
C declaration 210
COBOL declaration 210
PL/I declaration 210
Visual Basic declaration 210

Chapter 13. MQPMO - Put message options . . 211
Overview. 211
Fields 212
Context (MQHOBJ) 212
InvalidDestCount (MQLONG). 212
KnownDestCount (MQLONG) 212
Options (MQLONG) 213
PutMsgRecFields (MQLONG) 222
PutMsgRecOffset (MQLONG) 223
PutMsgRecPtr (MQPTR). 224
RecsPresent (MQLONG). 224
ResolvedQMgrName (MQCHAR48) 225
ResolvedQName (MQCHAR48) 225
ResponseRecOffset (MQLONG) 225
ResponseRecPtr (MQPTR) 226
StrucId (MQCHAR4) 227
Timeout (MQLONG) 227
UnknownDestCount (MQLONG). 227
Version (MQLONG) 227
Initial values and language declarations 228
C declaration 228
COBOL declaration 229

PL/I declaration 229
System/390 assembler declaration 230
TAL declaration 230
Visual Basic declaration 231

Chapter 14. MQPMR - Put-message record . . 233
Overview. 233
Fields 233
AccountingToken (MQBYTE32) 234
CorrelId (MQBYTE24) 234
Feedback (MQLONG) 234
GroupId (MQBYTE24) 234
MsgId (MQBYTE24) 235
Initial values and language declarations 235
C declaration 235
COBOL declaration 235
PL/I declaration 236
Visual Basic declaration 236

Chapter 15. MQRFH - Rules and formatting header 237
Overview. 237
Fields 237
CodedCharSetId (MQLONG) 237
Encoding (MQLONG) 238
Flags (MQLONG) 238
Format (MQCHAR8) 238
NameValueString (MQCHARn) 238
StrucId (MQCHAR4) 239
StrucLength (MQLONG) 239
Version (MQLONG) 240
Initial values and language declarations 240
C declaration 240
COBOL declaration 241
PL/I declaration 241
System/390 assembler declaration 241

Chapter 16. MQRFH2 - Rules and formatting header version 2 243
Overview. 243
Fields 244
CodedCharSetId (MQLONG) 244
Encoding (MQLONG) 244
Flags (MQLONG) 244
Format (MQCHAR8) 244
NameValueCCSID (MQLONG) 245
NameValueData (MQCHARn) 245
NameValueLength (MQLONG) 247
StrucId (MQCHAR4) 248
StrucLength (MQLONG) 248
Version (MQLONG) 248
Initial values and language declarations 249
C declaration 249
COBOL declaration 249
PL/I declaration 250
System/390 assembler declaration 250

Chapter 17. MQRMH - Reference message header 251
Overview. 251
Fields 252
CodedCharSetId (MQLONG) 252
DataLogicalLength (MQLONG) 253
DataLogicalOffset (MQLONG). 253
DataLogicalOffset2 (MQLONG) 254
DestEnvLength (MQLONG) 254
DestEnvOffset (MQLONG) 254
DestNameLength (MQLONG) 254
DestNameOffset (MQLONG) 254
Encoding (MQLONG) 255
Flags (MQLONG) 255
Format (MQCHAR8) 255
ObjectInstanceId (MQBYTE24). 256
ObjectType (MQCHAR8) 256
SrcEnvLength (MQLONG) 256
SrcEnvOffset (MQLONG) 256
SrcNameLength (MQLONG) 257
SrcNameOffset (MQLONG) 257
StrucId (MQCHAR4) 257
StrucLength (MQLONG) 257
Version (MQLONG) 258
Initial values and language declarations 258
C declaration 259
COBOL declaration 259
PL/I declaration 260
System/390 assembler declaration 260
Visual Basic declaration 261

Chapter 18. MQRR - Response record 263
Overview. 263
Fields 263
CompCode (MQLONG) 263
Reason (MQLONG) 263
Initial values and language declarations 264
C declaration 264
COBOL declaration 264
PL/I declaration 264
Visual Basic declaration 264

Chapter 19. MQTM - Trigger message 265
Overview. 265
Fields 267
ApplId (MQCHAR256) 267
ApplType (MQLONG) 267
EnvData (MQCHAR128). 268
ProcessName (MQCHAR48) 268
QName (MQCHAR48) 268
StrucId (MQCHAR4) 269
TriggerData (MQCHAR64) 269
UserData (MQCHAR128) 269
Version (MQLONG) 270
Initial values and language declarations 270
C declaration 270
COBOL declaration 271
PL/I declaration 271
System/390 assembler declaration 271
TAL declaration 271
Visual Basic declaration 272

Chapter 20. MQTMC2 - Trigger message 2 (character format) 273
Overview. 273

Fields 274
ApplId (MQCHAR256) 274
ApplType (MQCHAR4) 274
EnvData (MQCHAR128). 274
ProcessName (MQCHAR48) 274
QMgrName (MQCHAR48) 274
QName (MQCHAR48) 274
StrucId (MQCHAR4) 274
TriggerData (MQCHAR64) 274
UserData (MQCHAR128) 275
Version (MQCHAR4) 275
Initial values and language declarations 275
C declaration 275
COBOL declaration 276
PL/I declaration 276
System/390 assembler declaration 276
TAL declaration 277
Visual Basic declaration 277

Chapter 21. MQWIH - Work information header 279
Overview. 279
Fields 279
CodedCharSetId (MQLONG) 279
Encoding (MQLONG) 280
Flags (MQLONG) 280
Format (MQCHAR8) 280
MsgToken (MQBYTE16) 281
Reserved (MQCHAR32) 281
ServiceName (MQCHAR32) 281
ServiceStep (MQCHAR8) 281
StrucId (MQCHAR4) 281
StrucLength (MQLONG) 281
Version (MQLONG) 282
Initial values and language declarations 282
C declaration 282
COBOL declaration 283
PL/I declaration 283
System/390 assembler declaration 283

Chapter 22. MQXP - Exit parameter block (OS/390 only) 285
Overview. 285
Fields 285
ExitCommand (MQLONG) 285
ExitId (MQLONG). 286
ExitParmCount (MQLONG) 286
ExitReason (MQLONG) 286
ExitResponse (MQLONG) 287
ExitUserArea (MQBYTE16) 287
Reserved (MQLONG) 288
StrucId (MQCHAR4) 288
Version (MQLONG) 288
Language declarations 288
C declaration 288
COBOL declaration 288
PL/I declaration 289
System/390 assembler declaration 289

Chapter 23. MQXQH - Transmission queue header 291
Overview. 291
Fields 294
MsgDesc (MQMD1) 294
RemoteQMgrName (MQCHAR48) 294
RemoteQName (MQCHAR48) 294
StrucId (MQCHAR4) 295
Version (MQLONG) 295
Initial values and language declarations 295
C declaration 296
COBOL declaration 296
PL/I declaration 297
System/390 assembler declaration 298
TAL declaration 298
Visual Basic declaration 298

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 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:
- 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
- MQPTR – Pointer

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, or 40. 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.

© Copyright IBM Corp. 1994, 2000

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. MQSeries constants" on page 547.

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 7. MQGMO - Get-message options" on page 81 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. MQSeries constants" on page 547.

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.

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 AS/400, and a 4-byte boundary on other platforms.

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).

C declarations

Table 8. 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];`
MQHCONN	`typedef MQLONG MQHCONN;`
MQHOBJ	`typedef MQLONG MQHOBJ;`
MQLONG	`typedef long MQLONG;`
MQPTR	`typedef void MQPOINTER MQPTR;`

See “Data types” on page 17 for a description of the MQPOINTER macro variable.

COBOL declarations

Table 9. 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 (AIX, OS/2, OS/390, VSE/ESA, and Windows NT only)

Table 10. 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)`

Table 10. Elementary data types in PL/I (continued)

Data type	Representation
MQCHAR8	`char(8)`
MQCHAR12	`char(12)`
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 (OS/390 only)

Table 11. 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`

Table 11. Elementary data types in System/390 assembler (continued)

Data type	Representation
MQHOBJ	`DS F`
MQLONG	`DS F`
MQPTR	`DS F`

TAL declarations (Tandem NonStop Kernel only)

Table 12. 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 (Windows 3.1, Windows 95, Windows 98, and Windows NT)

Table 13. Elementary data types in Visual Basic

Data type	Representation
MQBYTE	`String*1`
MQBYTE8	`String*8`
MQBYTE16	`String*16`

Table 13. Elementary data types in Visual Basic (continued)

Data type	Representation
MQBYTE24	`String*24`
MQBYTE32	`String*32`
MQBYTE40	`String*40`
MQCHAR	`String*1`
MQCHAR4	`String*4`
MQCHAR8	`String*8`
MQCHAR12	`String*12`
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	`Long`

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 14. Structure data types used on MQI calls

Structure	Description	Calls where used
MQBO	Begin options	MQBEGIN
MQCNO	Connect options	MQCONNX
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

Table 15. 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 599, together with the associated data conversion calls.

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 AS/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 AS/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 AS/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.

Rules for structure data types

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 16.

Table 16. C header files

Filename	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.

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.

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").

The COPY files are summarized in Table 17. Note that not all of the files listed are available in all environments.

Table 17. COBOL COPY files

File name (with initial values)	File name (without initial values)	Contents
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

Table 17. COBOL COPY files (continued)

File name (with initial values)	File name (without initial values)	Contents
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
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
CMQTMV	CMQTML	Trigger message structure
CMQTMCV	CMQTMCL	Trigger message structure (character format)
CMQTMC2V	CMQTMC2L	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 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 AS/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 9. MQMD - Message descriptor" on page 125 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 discontiguous 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 18.

Table 18. PL/I INCLUDE files

Filename	Contents
CMQEPP	Entry points
CMQP	Structures and named constants

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(cmqepp);
```

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(cmqepp);

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 = '''MD  ''';

%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 19.

Table 19. Assembler macros

Filename	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

Table 19. Assembler macros (continued)

Filename	Contents
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
CMQTMC2A	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_"
```

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.

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 20.

Table 20. Visual Basic header files

File name	Contents
CMQB.BAS	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.

Data types

Chapter 2. MQBO - Begin options

The following table summarizes the fields in the structure.

Table 21. Fields in MQBO

Field	Description	Page
StrucId	Structure identifier	29
Version	Structure version number	30
Options	Options that control the action of MQBEGIN	29

Overview

Availability: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT; not available for MQSeries 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: Character data in MQBO must be in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute. Numeric data in MQBO must be in the native machine encoding; this is given by MQENC_NATIVE.

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.

© Copyright IBM Corp. 1994, 2000

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 22. Initial values of fields in MQBO

Field name	Name of constant	Value of constant
StrucId	MQBO_STRUC_ID	'BObb'
Version	MQBO_VERSION_1	1
Options	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 {
  MQCHAR4  StrucId;  /* Structure identifier */
  MQLONG   Version;  /* Structure version number */
  MQLONG   Options;  /* Options that control the action of MQBEGIN */
} MQBO;
```

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

```
dcl
 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      'Controls action of MQBEGIN'
End Type
```

MQBO - Language declarations

Chapter 3. MQCIH - CICS information header

The following table summarizes the fields in the structure.

Table 23. Fields in MQCIH

Field	Description	Page
StrucId	Structure identifier	43
Version	Structure version number	45
StrucLength	Length of MQCIH structure	44
Encoding	Reserved	37
CodedCharSetId	Reserved	37
Format	MQ format name of data that follows MQCIH	39
Flags	Flags	38
ReturnCode	Return code from bridge	42
CompCode	MQ completion code or CICS EIBRESP	37
Reason	MQ reason or feedback code, or CICS EIBRESP2	41
UOWControl	Unit-of-work control	45
GetWaitInterval	Wait interval for MQGET call issued by bridge task	40
LinkType	Link type	40
OutputDataLength	Output COMMAREA data length	40
FacilityKeepTime	Bridge facility release time	38
ADSDescriptor	Send/receive ADS descriptor	35
ConversationalTask	Whether task can be conversational	37
TaskEndStatus	Status at end of task	44
Facility	Bridge facility token	38
Function	MQ call name or CICS EIBFN function	39
AbendCode	Abend code	35
Authenticator	Password or passticket	36
Reserved1	Reserved	42
ReplyToFormat	MQ format name of reply message	41
RemoteSysId	Reserved	41
RemoteTransId	Reserved	41
TransactionId	Transaction to attach	44
FacilityLike	Terminal emulated attributes	38
AttentionId	AID key	36
StartCode	Transaction start code	43
CancelCode	Abend transaction code	36
NextTransactionId	Next transaction to attach	40
Reserved2	Reserved	42
Reserved3	Reserved	42

© Copyright IBM Corp. 1994, 2000

Table 23. Fields in MQCIH (continued)

Field	Description	Page
Note: The remaining fields are not present if *Version* is less than MQCIH_VERSION_2.		
CursorPosition	Cursor position	37
ErrorOffset	Offset of error in message	37
InputItem	Reserved	40
Reserved4	Reserved	42

Overview

Availability: AIX, HP-UX, OS/390, OS/2, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Purpose: The MQCIH structure describes the information that can be present at the start of a message sent to the CICS bridge through MQSeries for OS/390.

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.

Usage: If the values required by the application are the same as the initial values shown in Table 25 on page 46, and the bridge is running with AUTH=LOCAL or 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 24 shows which fields are set for different values of *ReturnCode*.

Table 24. Contents of error information fields in MQCIH structure

ReturnCode	*Function*	*CompCode*	*Reason*	*AbendCode*
MQCRC_OK	–	–	–	–
MQCRC_BRIDGE_ERROR	–	–	MQFB_CICS_*	–
MQCRC_MQ_API_ERROR MQCRC_BRIDGE_TIMEOUT	MQ call name	MQ *CompCode*	MQ *Reason*	–
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.

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 24 on page 35.

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.

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.

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 24 on page 35. The following values are possible when *Function* contains an MQ call name:

MQCFUNC_MQCONN
MQCONN call.

MQCFUNC_MQGET
MQGET call.

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 24 on page 35.

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.

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 24 on page 35). 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.

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).

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 25. Initial values of fields in MQCIH

Field name	Name of constant	Value of constant
StrucId	MQCIH_STRUC_ID	'CIHb'
Version	MQCIH_VERSION_2	2
StrucLength	MQCIH_LENGTH_2	180
Encoding	None	0
CodedCharSetId	None	0
Format	MQFMT_NONE	Blanks
Flags	MQCIH_NONE	0
ReturnCode	MQCRC_OK	0
CompCode	MQCC_OK	0
Reason	MQRC_NONE	0
UOWControl	MQCUOWC_ONLY	273
GetWaitInterval	MQCGWI_DEFAULT	-2
LinkType	MQCLT_PROGRAM	1
OutputDataLength	MQCODL_AS_INPUT	-1
FacilityKeepTime	None	0
ADSDescriptor	MQCADSD_NONE	0
ConversationalTask	MQCCT_NO	0
TaskEndStatus	MQCTES_NOSYNC	0
Facility	MQCFAC_NONE	Nulls
Function	MQCFUNC_NONE	Blanks
AbendCode	None	Blanks
Authenticator	None	Blanks
Reserved1	None	Blanks
ReplyToFormat	MQFMT_NONE	Blanks
RemoteSysId	None	Blanks
RemoteTransId	None	Blanks
TransactionId	None	Blanks
FacilityLike	None	Blanks
AttentionId	None	Blanks
StartCode	MQCSC_NONE	Blanks
CancelCode	None	Blanks
NextTransactionId	None	Blanks

Table 25. Initial values of fields in MQCIH (continued)

Field name	Name of constant	Value of constant
Reserved2	None	Blanks
Reserved3	None	Blanks
CursorPosition	None	0
ErrorOffset	None	0
InputItem	None	0
Reserved4	None	0

Notes:

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:

   ```
   MQCIH MyCIH = {MQCIH_DEFAULT};
   ```

C declaration

```
typedef 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 */
  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 */
} MQCIH;
```

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
   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-REMOTETRANSID      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

```
dcl
 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 conversa-
                                         tional */
  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 */
  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_FACILITYKEEPTIME  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  Length of structure
                        ORG  MQCIH
MQCIH_AREA              DS   CL(MQCIH_LENGTH)
```

Chapter 4. MQCNO - Connect options

The following table summarizes the fields in the structure.

Table 26. Fields in MQCNO

Field	Description	Page
StrucId	Structure identifier	57
Version	Structure version number	57
Options	Options that control the action of MQCONNX	54
Note: The remaining fields are ignored if *Version* is less than MQCNO_VERSION_2.		
ClientConnOffset	Offset of MQCD structure for client connection	52
ClientConnPtr	Address of MQCD structure for client connection	52
Note: The remaining fields are ignored if *Version* is less than MQCNO_VERSION_3.		
ConnTag	Queue-manager connection tag.	54

Overview

Availability:
- Versions 1 and 2: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems
- Version 3: OS/390 only

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 MQCONNX call.

Version: The current version of MQCNO is MQCNO_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 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.

Character set and encoding: Character data in MQCNO must be in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute. Numeric data in MQCNO must be in the native machine encoding; this is given by MQENC_NATIVE.

Fields

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

© Copyright IBM Corp. 1994, 2000

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 MQCONNX call is running as an 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.

ClientConnPtr (MQPTR)

Address of MQCD structure for client connection.

ClientConnOffset and *ClientConnPtr* are used only when the application issuing the MQCONNX call is running as an 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 an 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 an 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 *MQSeries Intercommunication* book for more information about the fields in the MQCD structure.

Field in MQCD	Value
ChannelName	Channel name.
Version	Structure version number. Must not be less than MQCD_VERSION_6.
TransportType	Any supported transport type.
ModeName	LU 6.2 mode name.
TpName	LU 6.2 transaction program name.
SecurityExit	Name of channel security exit.
SendExit	Name of channel send exit.
ReceiveExit	Name of channel receive exit.
MaxMsgLength	Maximum length in bytes of messages that can be sent over the client connection channel.
SecurityUserData	User data for security exit.
SendUserData	User data for send exit.
ReceiveUserData	User data for receive exit.
UserIdentifier	User identifier to be used to establish an LU 6.2 session.
Password	Password to be used to establish an LU 6.2 session.
ConnectionName	Connection name.
HeartbeatInterval	Time in seconds between heartbeat flows.
StrucLength	Length of the MQCD structure.

Field in MQCD	Value
ExitNameLength	Length of exit names addressed by *SendExitPtr* and *ReceiveExitPtr*. Must be greater than zero if *SendExitPtr* or *ReceiveExitPtr* is set to a value that is not the null pointer.
ExitDataLength	Length of exit data addressed by *SendUserDataPtr* and *ReceiveUserDataPtr*. Must be greater than zero if *SendUserDataPtr* or *ReceiveUserDataPtr* is set to a value that is not the null pointer.
SendExitsDefined	Number of send exits addressed by *SendExitPtr*. If zero, *SendExit* and *SendUserData* provide the exit name and data. If greater than zero, *SendExitPtr* and *SendUserDataPtr* provide the exit names and data, and *SendExit* and *SendUserData* must be blank.
ReceiveExitsDefined	Number of receive exits addressed by *ReceiveExitPtr*. If zero, *ReceiveExit* and *ReceiveUserData* provide the exit name and data. If greater than zero, *ReceiveExitPtr* and *ReceiveUserDataPtr* provide the exit names and data, and *ReceiveExit* and *ReceiveUserData* must be blank.
SendExitPtr	Address of name of first send exit.
SendUserDataPtr	Address of data for first send exit.
ReceiveExitPtr	Address of name of first receive exit.
ReceiveUserDataPtr	Address of data for first receive exit.
LongRemoteUserIdLength	Length of long remote user identifier.
LongRemoteUserIdPtr	Address of long remote user identifier.
RemoteSecurityId	Remote security identifier.

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 its own 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.

  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).
- 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).

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 called has completed, the MQCD structure is not referenced again.

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.

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.

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.

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.

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

MQCT_NONE
No connection tag specified.

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 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 MQCONNX.

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 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 *MQSeries Application Programming Guide* for more information.
- The application must not have more than one thread connected to the queue manager at any one time.
- 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 AS/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 program must run with the `mqm` user identifier and the `mqm` group identifier. 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.
- On Windows NT, the program must be a member of the `mqm` group.

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

This option is supported in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT. On OS/390 the option is accepted but ignored.

On AIX, HP-UX, OS/2, Sun Solaris, and Windows NT, 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:

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 OS/390, 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.
- 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 OS/390, 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.

These options are supported only on OS/390.

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.

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 only on OS/390.

MQCNO - Fields

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 27. Initial values of fields in MQCNO

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

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 {
  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 */
} MQCNO;
```

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 MQCONNX
   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).
```

PL/I declaration

```
dcl
 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 MQCONNX */
  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 */
```

System/390 assembler declaration (OS/390)

```
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 MQCONNX
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  Length of structure
                               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'
  Options           As Long      'Controls action of MQCONNX'
  ClientConnOffset  As Long      'Offset of MQCD structure for client'
                                 'connection'
  ClientConnPtr     As String*32 'Address of MQCD structure for client'
                                 'connection'
End Type
```

Note: The *ClientConnPtr* field is not used, and is set to 32 null characters by default.

MQCNO - Language declarations

Chapter 5. MQDH - Distribution header

The following table summarizes the fields in the structure.

Table 28. Fields in MQDH

Field	Description	Page
StrucId	Structure identifier	65
Version	Structure version number	66
StrucLength	Length of MQDH structure plus following records	65
Encoding	Numeric encoding of data that follows array of MQPMR records	63
CodedCharSetId	Character set identifier of data that follows array of MQPMR records	62
Format	Format name of data that follows array of MQPMR records	64
Flags	General flags	63
PutMsgRecFields	Flags indicating which MQPMR fields are present	64
RecsPresent	Number of object records present	65
ObjectRecOffset	Offset of first object record from start of MQDH	64
PutMsgRecOffset	Offset of first put-message record from start of MQDH	65

Overview

Availability: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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.

Character set and encoding: Character data in MQDH must be in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute. Numeric data in MQDH must be in the native machine encoding; this is given by the value of MQENC_NATIVE for the C programming language.

© Copyright IBM Corp. 1994, 2000

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 39. Attributes for queues" on page 427). 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 array of 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.

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.

This value is supported in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG)

Numeric encoding of data that follows array of 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.

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 array of 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 object 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.

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 put message 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 object 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 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 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 29. Initial values of fields in MQDH

Field name	Name of constant	Value of constant
StrucId	MQDH_STRUC_ID	'DHbb'
Version	MQDH_VERSION_1	1
StrucLength	None	0
Encoding	None	0
CodedCharSetId	MQCCSI_UNDEFINED	0
Format	MQFMT_NONE	Blanks
Flags	MQDHF_NONE	0
PutMsgRecFields	MQPMRF_NONE	0
RecsPresent	None	0
ObjectRecOffset	None	0
PutMsgRecOffset	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 {
  MQCHAR4  StrucId;          /* Structure identifier */
  MQLONG   Version;          /* Structure version number */
  MQLONG   StrucLength;      /* Length of MQDH structure plus following
                                records */
  MQLONG   Encoding;         /* Numeric encoding of data that follows
                                array of MQPMR records */
  MQLONG   CodedCharSetId;   /* Character set identifier of data that
                                follows array of MQPMR records */
  MQCHAR8  Format;           /* Format name of data that follows array
                                of MQPMR records */
  MQLONG   Flags;            /* General flags */
```

```
  MQLONG   PutMsgRecFields;  /* Flags indicating which MQPMR fields are
                                present */
  MQLONG   RecsPresent;      /* Number of object records present */
  MQLONG   ObjectRecOffset;  /* Offset of first object record from start
                                of MQDH */
  MQLONG   PutMsgRecOffset;  /* Offset of first put message record from
                                start of MQDH */
} 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 records
   15 MQDH-STRUCLENGTH     PIC S9(9) BINARY.
**    Numeric encoding of data that follows array of MQPMR records
   15 MQDH-ENCODING        PIC S9(9) BINARY.
**    Character set identifier of data that follows array of MQPMR
**    records
   15 MQDH-CODEDCHARSETID  PIC S9(9) BINARY.
**    Format name of data that follows array of MQPMR records
   15 MQDH-FORMAT          PIC X(8).
**    General flags
   15 MQDH-FLAGS           PIC S9(9) BINARY.
**    Flags indicating which MQPMR fields are present
   15 MQDH-PUTMSGRECFIELDS PIC S9(9) BINARY.
**    Number of object records present
   15 MQDH-RECSPRESENT     PIC S9(9) BINARY.
**    Offset of first object record from start of MQDH
   15 MQDH-OBJECTRECOFFSET PIC S9(9) BINARY.
**    Offset of first put message record from start of MQDH
   15 MQDH-PUTMSGRECOFFSET PIC S9(9) BINARY.
```

PL/I declaration

```
dcl
 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 fol-
                                      lowing records */
  3 Encoding        fixed bin(31), /* Numeric encoding of data that
                                      follows array of MQPMR records */
  3 CodedCharSetId  fixed bin(31), /* Character set identifier of data
                                      that follows array of MQPMR
                                      records */
  3 Format          char(8),       /* Format name of data that follows
                                      array of MQPMR records */
  3 Flags           fixed bin(31), /* General flags */
  3 PutMsgRecFields fixed bin(31), /* Flags indicating which MQPMR
                                      fields are present */
  3 RecsPresent     fixed bin(31), /* Number of object records
                                      present */
  3 ObjectRecOffset fixed bin(31), /* Offset of first object record from
                                      start of MQDH */
  3 PutMsgRecOffset fixed bin(31); /* Offset of first put message record
                                      from start of MQDH */
```

Visual Basic declaration

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

MQDH - Language declarations

```
  StrucLength       As Long      'Length of MQDH structure plus'
                                 'following records'
  Encoding          As Long      'Encoding of message data'
  CodedCharSetId    As Long      'Coded character-set identifier'
                                 'of message data'
  Format            As String*8  'Format name of message data'
  Flags             As Long      'Format name of message data'
  PutMsgRecFields   As Long      'Flags indicating which MQPMR fields'
                                 'are present'
  RecsPresent       As Long      'Number of object records present'
  ObjectRecOffset   As Long      'Offset of first object record from'
                                 'start of MQDH'
  PutMsgRecOffset   As Long      'Offset of first put message record'
                                 'from start of MQDH'
End Type
```

Chapter 6. MQDLH - Dead-letter header

The following table summarizes the fields in the structure.

Table 30. Fields in MQDLH

Field	Description	Page
StrucId	Structure identifier	75
Version	Structure version number	75
Reason	Reason message arrived on dead-letter queue	74
DestQName	Name of original destination queue	72
DestQMgrName	Name of original destination queue manager	71
Encoding	Numeric encoding of data that follows MQDLH	72
CodedCharSetId	Character set identifier of data that follows MQDLH	71
Format	Format name of data that follows MQDLH	72
PutApplType	Type of application that put message on dead-letter queue	73
PutApplName	Name of application that put message on dead-letter queue	72
PutDate	Date when message was put on dead-letter queue	73
PutTime	Time when message was put on dead-letter queue	73

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.

© Copyright IBM Corp. 1994, 2000

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.

This value is supported in the following environments: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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.

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.

PutApplName (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 *PutApplName* field in "Chapter 9. MQMD - Message descriptor" on page 125.

If it is the queue manager that redirects the message to the dead-letter queue, *PutApplName* 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.

PutApplType (MQLONG)

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

This field has the same meaning as the *PutApplType* field in the message descriptor MQMD (see "Chapter 9. MQMD - Message descriptor" on page 125 for details).

If it is the queue manager that redirects the message to the dead-letter queue, *PutApplType* 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 *MQSeries System Administration*.

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.

On OS/2, the queue manager uses the TZ environment variable to calculate GMT. For more information on setting this variable, refer to the *MQSeries System Administration* 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 9. MQMD - Message descriptor" on page 125 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 *ApplId* field of the trigger message (see "Chapter 19. MQTM - Trigger message" on page 265).

On OS/390, 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 *ApplType* field of the trigger message is not valid (see "Chapter 19. MQTM - Trigger message" on page 265).

On OS/390, 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 OS/390, 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 31. Initial values of fields in MQDLH

Field name	Name of constant	Value of constant
StrucId	MQDLH_STRUC_ID	'DLHb'
Version	MQDLH_VERSION_1	1
Reason	MQRC_NONE	0
DestQName	None	Null string or blanks
DestQMgrName	None	Null string or blanks
Encoding	None	0
CodedCharSetId	MQCCSI_UNDEFINED	0
Format	MQFMT_NONE	Blanks
PutApplType	None	0
PutApplName	None	Null string or blanks
PutDate	None	Null string or blanks
PutTime	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 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:

   ```
   MQDLH MyDLH = {MQDLH_DEFAULT};
   ```

C declaration

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

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

```
dcl
 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 PutApplType    fixed bin(31), /* Type of application that put
                                     message on dead-letter
                                     (undelivered-message) queue */
  3 PutApplName    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 */
  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  Length of structure
                        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'
  Encoding             As Long      'Original data encoding'
  CodedCharSetId       As Long      'Original coded character set identifier'
  Format               As String*8  'Original format name'
  PutApplType          As Long      'Type of application that put the'
                                    'message on dead-letter'
                                    '(undelivered-message) queue'
  PutApplName          As String*28 'Name of application that put the'
                                    'message on dead-letter'
                                    '(undelivered-message) queue'
  PutDate              As String*8  'Date when message was put on the'
                                    '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 7. MQGMO - Get-message options

The following table summarizes the fields in the structure.

Table 32. Fields in MQGMO

Field	Description	Page
StrucId	Structure identifier	111
Version	Structure version number	111
Options	Options that control the action of MQGET	86
WaitInterval	Wait interval	112
Signal1	Signal	110
Signal2	Signal identifier	111
ResolvedQName	Resolved name of destination queue	108
Note: The remaining fields are ignored if *Version* is less than MQGMO_VERSION_2.		
MatchOptions	Options controlling selection criteria used for MQGET	82
GroupStatus	Flag indicating whether message retrieved is in a group	82
SegmentStatus	Flag indicating whether message retrieved is a segment of a logical message	109
Segmentation	Flag indicating whether further segmentation is allowed for the message retrieved	109
Reserved1	Reserved	108
Note: The remaining fields are ignored if *Version* is less than MQGMO_VERSION_3.		
MsgToken	Message token	85
ReturnedLength	Length of message data returned (bytes)	109

Overview

Availability:
- Version 1: All
- Versions 2 and 3: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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.

The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQGMO that is supported by the

© Copyright IBM Corp. 1994, 2000

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: Character data in MQGMO must be in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute. Numeric data in MQGMO must be in the native machine encoding; this is given by MQENC_NATIVE.

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.

On OS/390, the queue manager always sets this field to MQGS_NOT_IN_GROUP.

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 OS/390, 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 is 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).

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.

This option is not supported on OS/390.

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.

This option is not supported on OS/390.

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 OS/390.

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 OS/390.

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_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: OS/390.

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.

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:

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 OS/390, 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.
- 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.

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. The MQGMO_SET_SIGNAL option is environment specific, and should not be used by applications which 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 OS/390, the signal is delivered by posting the ECB.
- On Tandem NonStop Kernel, an inter-process communication (IPC) message is sent to the $RECEIVE queue of the process issuing the MQGET call.
- On Windows 95, Windows 98, a Windows message is sent to the application.

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.

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 given handle can have no more than one signal outstanding.

This option is not valid with any of the following options:
MQGMO_UNLOCK

MQGMO_WAIT

This option is supported in the following environments: OS/390, Tandem 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 OS/390, 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 OS/390, 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 Windows 3.1, Windows 95, Windows 98, this option is accepted but ignored.

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 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 OS/390, Tandem 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, OS/390, OS/2, AS/400, Sun Solaris, Windows 95, Windows 98, Windows NT, plus MQSeries 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 OS/390, Tandem 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 your application, but only after some resource updates have been made does it become apparent that the unit of work cannot complete successfully. A normal backout, if this option had not been specified, would cause the message to be reinstated on the queue, so that the same sequence of events would occur when the message was next retrieved. Using this option on the original MQGET, however, means that the backout will cause the updates to the other resources to be backed out, as is required, but the message is treated as if it had been retrieved under a new unit of work. The application can now perform some exception handling, such as informing the originator that the message has been discarded, and commit this new unit of work, which causes the message to be removed from the queue.

This option has an effect only if the unit of work containing the get request is terminated by an application request to back 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, the abend of a transaction or the system). In this situation, any message retrieved using this option is backed out on to 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, returning a message obtained with the MQGMO_MARK_SKIP_BACKOUT option to the queue, must issue an MQ call (any MQ call will do) in between the two ROLB commands.
2. A CICS application, returning a message obtained with the MQGMO_MARK_SKIP_BACKOUT option to the queue, must issue an MQ call (any MQ call will do) in between the two EXEC CICS SYNCPOINT ROLLBACK commands.

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 OS/390.

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 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 39. Attributes for queues" on page 427 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 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 OS/390 and 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).

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: OS/390, 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.

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.

This option is not supported in the following environments: OS/390, Windows 3.1, Windows 95, Windows 98.

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 599 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 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 9. MQMD - Message descriptor" on page 125 for a list of format names for which the queue manager performs the conversion.

This option is not supported in the following environments: OS/390 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 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 7. MQGMO - Get-message options" on page 81 for further details.

Table 33 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:

LOG ORD
A "✓" means that the row applies only when the MQGMO_LOGICAL_ORDER option is specified.

Cur grp
A "✓" means that the row applies only when a current message group exists prior to the call.

A "(✓)" means that the row applies whether or not a current message group exists prior to the call.

Cur log msg
A "✓" means that the row applies only when a current logical message exists prior to the call.

A "(✓)" means that the row applies whether or not a current logical message exists prior to the call.

Other columns
These 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 33. 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				
LOG ORD	Cur grp	Cur log msg	*MsgId*	*CorrelId*	*GroupId*	*MsgSeqNumber*	*Offset*
✓			Controlled by *MatchOptions*	Controlled by *MatchOptions*	Controlled by *MatchOptions*	1	0

Table 33. MQGET options relating to messages in groups and segments of logical messages (continued)

Options you specify	Group and log-msg status prior to call		Values the queue manager looks for				
✓		✓	Any message identifier	Any correlation identifier	Previous group identifier	1	Previous offset + previous segment length
✓	✓		Any message identifier	Any correlation identifier	Previous group identifier	Previous sequence number + 1	0
✓	✓	✓	Any message identifier	Any correlation identifier	Previous group identifier	Previous sequence number	Previous offset + previous segment length
	(✓)	(✓)	Controlled by *MatchOptions*	Controlled by *MatchOptions*	Controlled by *MatchOptions*	Controlled by *MatchOptions*	Controlled by *MatchOptions*

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 7. MQGMO - Get-message options" on page 81 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.

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:

- Each successful MQGET call that does *not* specify MQGMO_LOGICAL_ORDER 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, but the message or segment retrieved is not the next one in the group or logical message. The call may however succeed with an MQCC_WARNING completion code. Table 34 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 34. Outcome when MQGET or MQCLOSE call is not consistent with group and segment information

Current call	Previous call	
	MQGET with MQGMO_LOGICAL_ORDER	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).

This option is supported in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

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.
- 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_CCSIDS 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.

This option is supported in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

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.

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.

This option is supported in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

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.

This option is supported in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

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:

- 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 OS/390, 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 OS/390, 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 OS/390, 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. 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 OS/390, 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 OS/390 mapping macro IHAECB as being for a user completion code.
- On Tandem 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.
- On Windows 95, Windows 98, this field must contain the window handle of the window to which the signal is to be sent. If this is zero, the signal is sent to the thread requesting the signal. The signal is a Windows message with the identifier specified by the *Signal2* field. The message contains a signal completion code in the WPARAM field.
- 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.

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 OS/390, 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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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 9. MQMD - Message descriptor" on page 125 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 OS/390, 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 35. Initial values of fields in MQGMO

Field name	Name of constant	Value of constant
StrucId	MQGMO_STRUC_ID	'GMOb'
Version	MQGMO_VERSION_1	1
Options	MQGMO_NO_WAIT	0
WaitInterval	None	0
Signal1	None	Null pointer on OS/390; 0 otherwise
Signal2	None	0
ResolvedQName	None	Null string or blanks
MatchOptions	MQMO_MATCH_MSG_ID + MQMO_MATCH_CORREL_ID	3
GroupStatus	MQGS_NOT_IN_GROUP	'b'
SegmentStatus	MQSS_NOT_A_SEGMENT	'b'
Segmentation	MQSEG_INHIBITED	'b'
Reserved1	None	'b'
MsgToken	MQMTOK_NONE	Nulls
ReturnedLength	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 {
  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 segmen-
                                tation is allowed for the message
                                retrieved */
  MQCHAR    Reserved1;       /* Reserved */
```

```
  MQBYTE16  MsgToken;        /* Message token */
  MQLONG    ReturnedLength;  /* Length of message data returned
                                (bytes) */
} MQGMO;
```

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.
```

PL/I declaration

```
dcl
 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 cri-
                                     teria 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 */
  3 Reserved1      char(1),       /* Reserved */
  3 MsgToken       char(16),      /* Message token */
  3 ReturnedLength fixed bin(31); /* Length of message data returned
                                     (bytes) */
```

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  Length of structure
                        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 message 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'
                             'retrieved is a segment of a logical message'
```

MQGMO - Language declarations

```
  Segmentation  As String*1  'Flag indicating whether further segmentation'
                             'is allowed for the message retrieved'
  Reserved1     As String*1  'Reserved'
End Type
```

Chapter 8. MQIIH - IMS information header

The following table summarizes the fields in the structure.

Table 36. Fields in MQIIH

Field	Description	Page
StrucId	Structure identifier	120
Version	Structure version number	121
StrucLength	Length of MQIIH structure	120
Encoding	Reserved	119
CodedCharSetId	Reserved	118
Format	MQ format name of data that follows MQIIH	119
Flags	Flags	119
LTermOverride	Logical terminal override	119
MFSMapName	Message format services map name	119
ReplyToFormat	MQ format name of reply message	119
Authenticator	RACF™ password or passticket	118
TranInstanceId	Transaction instance identifier	121
TranState	Transaction state	121
CommitMode	Commit mode	118
SecurityScope	Security scope	120
Reserved	Reserved	120

Overview

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

Purpose: The MQIIH structure describes the information that must be present at the start of a message sent to the IMS bridge through MQSeries for OS/390.

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.

© Copyright IBM Corp. 1994, 2000

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.

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.

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.

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.

TranInstanceId (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 *MQSeries Application Programming Guide* for further details.

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 37. Initial values of fields in MQIIH

Field name	Name of constant	Value of constant
StrucId	MQIIH_STRUC_ID	'IIHb'
Version	MQIIH_VERSION_1	1
StrucLength	MQIIH_LENGTH_1	84
Encoding	None	0
CodedCharSetId	None	0
Format	MQFMT_NONE	Blanks
Flags	MQIIH_NONE	0
LTermOverride	None	Blanks
MFSMapName	None	Blanks
ReplyToFormat	MQFMT_NONE	Blanks
Authenticator	MQIAUT_NONE	Blanks
TranInstanceId	MQITII_NONE	Nulls
TranState	MQITS_NOT_IN_CONVERSATION	'b'
CommitMode	MQICM_COMMIT_THEN_SEND	'0'
SecurityScope	MQISS_CHECK	'C'
Reserved	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 {
  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   MFSMapName;      /* 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 */
} MQIIH;
```

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-MFSMAPNAME     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 */
  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 MFSMapName     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_STRUCLENGTH       DS   F        Length of MQIIH structure
MQIIH_ENCODING          DS   F        Reserved
MQIIH_CODEDCHARSETID    DS   F        Reserved
```

MQIIH - Language declarations

```
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_MFSMAPNAME            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  Length of structure
                            ORG  MQIIH
MQIIH_AREA                  DS   CL(MQIIH_LENGTH)
```

Chapter 9. MQMD - Message descriptor

The following table summarizes the fields in the structure.

Table 38. Fields in MQMD

Field	Description	Page
StrucId	Structure identifier	176
Version	Structure version number	178
Report	Options for report messages	165
MsgType	Message type	154
Expiry	Message lifetime	134
Feedback	Feedback or reason code	136
Encoding	Numeric encoding of message data	133
CodedCharSetId	Character set identifier of message data	131
Format	Format name of message data	140
Priority	Message priority	158
Persistence	Message persistence	156
MsgId	Message identifier	151
CorrelId	Correlation identifier	132
BackoutCount	Backout counter	130
ReplyToQ	Name of reply queue	164
ReplyToQMgr	Name of reply queue manager	165
UserIdentifier	User identifier	176
AccountingToken	Accounting token	127
ApplIdentityData	Application data relating to identity	129
PutApplType	Type of application that put the message	160
PutApplName	Name of application that put the message	159
PutDate	Date when message was put	162
PutTime	Time when message was put	163
ApplOriginData	Application data relating to origin	130
Note: The remaining fields are ignored if *Version* is less than MQMD_VERSION_2.		
GroupId	Group identifier	146
MsgSeqNumber	Sequence number of logical message within group	153
Offset	Offset of data in physical message from start of logical message	155
MsgFlags	Message flags	147
OriginalLength	Length of original message	156

Overview

Availability:
- Version 1: All

© Copyright IBM Corp. 1994, 2000

- Version 2: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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: Character data in MQMD must be in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute. Numeric data in MQMD must be in the native machine encoding; this is given by MQENC_NATIVE.

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 follows:

- 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.

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 42 on page 190.

When a message is on a transmission queue, some of the fields in MQMD are set to particular values; see "Chapter 23. MQXQH - Transmission queue header" on page 291 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 *MQSeries 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 125; also see the *MQSeries 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 OS/390 the accounting information is set to:
    - For OS/390 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 AS/400, the accounting information is set to the accounting code for the job.
  - On Compaq (DIGITAL) OpenVMS, Tandem NonStop Kernel, and UNIX systems, the accounting information is set to the numeric user identifier, in ASCII characters.
  - On OS/2, DOS client, Windows client, Windows 3.1, and Windows 95, Windows 98, the accounting information is set to the ASCII character '1'.
  - On Windows NT, 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
  DOS client default accounting token.

  MQACTT_NT_SECURITY_ID
  Windows NT security identifier.

  MQACTT_OS2_DEFAULT
  OS/2 default accounting token.

  MQACTT_OS400_ACCOUNT_TOKEN
  AS/400 accounting token.

  MQACTT_UNIX_NUMERIC_ID
  UNIX systems numeric identifier.

  MQACTT_WINDOWS_DEFAULT
  Windows client, Windows 3.1, or 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, AS/400, Sun Solaris, Windows NT, plus MQSeries 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 *MQSeries 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.

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 125; also see the *MQSeries 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 *MQSeries 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 125; also see the *MQSeries 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 39. Attributes for queues" on page 427.

On OS/390, 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 42. Attributes for the queue manager" on page 469 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_INHERIT
Inherit character-set identifier of this structure.

Character data in the message is in the same character set as MQMD; this is the queue-manager's character set. When MQCCSI_INHERIT is specified in MQMD, it 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.

This value is supported in the following environments: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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 identifier 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.

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 589 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 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.

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:

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: 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 MQSeries 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.

CICS-bridge feedback codes: 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 489.

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.

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 *MQSeries Programmable System Management* 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.

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 OS/390, 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 6. MQDLH - Dead-letter header" on page 69 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 5. MQDH - Distribution header" on page 61 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, AS/400, Sun Solaris, Windows NT, plus MQSeries 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.

The message is an MQ event message that reports an event that occurred. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call. Event messages have the same structure as programmable commands; Refer to the *MQSeries Programmable System Management* book for more information about this structure, and to the *MQSeries Event Monitoring* book for information about events.

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.

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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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.

The message is an IMS variable string, which is a string of the form `llzzccc`, where:

ll is a 2-byte length field specifying the total length of the IMS variable string item. This length is equal to the length of `ll` (2 bytes), plus the length of `zz` (2 bytes), plus the length of the character string itself. `ll` 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 `ll-4` characters. `ccc` is in the character set specified by the *CodedCharSetId* field.

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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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 10. MQMDE - Message descriptor extension" on page 185 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, AS/400, Sun Solaris, Windows NT, plus MQSeries 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 *MQSeries Programmable System Management* book for more information about using programmable command format messages.

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 17. MQRMH - Reference message header" on page 251 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, AS/400, Sun Solaris, Windows NT, plus MQSeries 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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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 19. MQTM - Trigger message" on page 265 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 OS/390, 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.

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 23. MQXQH - Transmission queue header" on page 291 for details of this structure.

COA and COD reports are not generated for messages which have a *Format* of MQFMT_XMIT_Q_HEADER.

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 48 on page 217. 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 33 on page 101. 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 7. MQGMO - Get-message options" on page 81 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.

The value of this flag in 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.

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 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.

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.

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 48 on page 217 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 593.

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

identifier [3] 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.

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.

3. 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 MQSeries 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.

- On OS/390, 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 33 on page 101 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.

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 48 on page 217. 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 33 on page 101. 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.

On input to the MQPUT and MQPUT1 calls, the queue manager uses the value detailed in Table 48 on page 217. 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 33 on page 101. 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

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.

  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.

When replying to a message, applications should normally use for the reply message the priority of the request message. In other situations, defaulting to the queue definition allows priority tuning to be carried out without changing the application.

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.

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.

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.

PutApplName (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 125; also see the *MQSeries Application Programming Guide*.

The format of the *PutApplName* 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 OS/390, the queue manager uses:
  - For OS/390 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
  - 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, DOS client, Windows client, and Windows NT, 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 AS/400, the queue manager uses the fully-qualified job name.
- On Compaq (DIGITAL) OpenVMS and Tandem 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 *PutApplName* 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.

PutApplType (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 125; also see the *MQSeries Application Programming Guide*.

PutApplType 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_CICS
CICS transaction.

MQAT_CICS_BRIDGE
CICS bridge.

MQAT_CICS_VSE
CICS/VSE® transaction.

MQAT_DOS
DOS client application.

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
OS/390 or TSO application (same value as MQAT_OS390).

MQAT_NOTES_AGENT
Lotus® Notes™ Agent application.

MQAT_NSK
Tandem NonStop Kernel application.

MQAT_OS2
OS/2 or Presentation Manager® application.

MQAT_OS390
OS/390 application.

MQAT_OS400
AS/400 application.

MQAT_QMGR
Queue manager.

MQAT_UNIX
UNIX application.

MQAT_VMS
Digital OpenVMS application.

MQAT_VOS
Stratus VOS application.

MQAT_WINDOWS
Windows client, Windows 3.1 application.

MQAT_WINDOWS_NT
Windows NT, Windows 95, Windows 98 application.

MQAT_XCF
XCF.

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 AS/400, it is set to MQAT_OS400; the queue manager never uses MQAT_CICS on AS/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 125; also see the *MQSeries 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 *MQSeries System Administration* 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 125; also see the *MQSeries Application Programming Guide*.

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.

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 42. Attributes for the queue manager” on page 469).

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 *MQSeries Application Programming Guide*.

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 *MQSeries 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 length of the message data in the original message is less than 100 bytes, the length of the message data in the report is the same length as the original message.

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.

Do not specify more than one of MQRO_EXCEPTION, MQRO_EXCEPTION_WITH_DATA, and MQRO_EXCEPTION_WITH_FULL_DATA.

On OS/390, the MQRO_EXCEPTION_WITH_FULL_DATA option is not supported.

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 length of the message data in the original message is less than 100 bytes, the length of the message data in the report is the same length as the original message.

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.

On OS/390, the MQRO_EXPIRATION_WITH_FULL_DATA option is not supported.

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 length of the message data in the original message is less than 100 bytes, the length of the message data in the report is the same length as the original message.

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.

On OS/390, the MQRO_COA_WITH_FULL_DATA option is not supported.

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 report message. If the length of the message data in the original message is less than 100 bytes, the length of the message data in the report is the same length as the original message.

If MQGMO_ACCEPT_TRUNCATED_MSG is specified on the MQGET call for the original message, and the message returned is truncated, the amount of message data placed in the report message depends on the environment:

- On OS/390, 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.

On OS/390, the MQRO_COD_WITH_FULL_DATA option is not supported.

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.

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.

On OS/390, the MQRO_DISCARD_MSG option is not supported.

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: 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 593 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.

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.

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.

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.

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:

1. The report message is held up because a link is down.
2. 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).
3. The report message is on a dead-letter queue.
4. 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.
5. 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.

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: 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.

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.

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
StrucId	MQMD_STRUC_ID
Version	MQMD_VERSION_2
Report	MQRO_NONE
MsgType	MQMT_REPORT
Expiry	MQEI_UNLIMITED
Feedback	As appropriate for the nature of the report (MQFB_COA, MQFB_COD, MQFB_EXPIRATION, or an MQRC_* value)
Encoding	Copied from the original message descriptor
CodedCharSetId	Copied from the original message descriptor
Format	Copied from the original message descriptor
Priority	Copied from the original message descriptor
Persistence	Copied from the original message descriptor
MsgId	As specified by the report options in the original message descriptor
CorrelId	As specified by the report options in the original message descriptor
BackoutCount	0
ReplyToQ	Blanks
ReplyToQMgr	Name of queue manager
UserIdentifier	As set by the MQPMO_PASS_IDENTITY_CONTEXT option
AccountingToken	As set by the MQPMO_PASS_IDENTITY_CONTEXT option
ApplIdentityData	As set by the MQPMO_PASS_IDENTITY_CONTEXT option
PutApplType	MQAT_QMGR, or as appropriate for the message channel agent
PutApplName	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.
PutDate	Date when report message is sent
PutTime	Time when report message is sent
ApplOriginData	Blanks
GroupId	Copied from the original message descriptor
MsgSeqNumber	Copied from the original message descriptor
Offset	Copied from the original message descriptor
MsgFlags	Copied from the original message descriptor
OriginalLength	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 594.

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 125; also see the *MQSeries 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 OS/390, 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 OS/390, 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 AS/400, the queue manager uses the name of the user profile associated with the application job.
- On Tandem NonStop Kernel, the queue manager uses the MQSeries principal that is defined for the Tandem user identifier in the MQSeries principal database.
- On Compaq (DIGITAL) OpenVMS 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 NT, the queue manager uses the first 12 characters of the logged-on user name.

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, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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 402 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 39. Initial values of fields in MQMD

Field name	Name of constant	Value of constant
StrucId	MQMD_STRUC_ID	'MDbb'
Version	MQMD_VERSION_1	1
Report	MQRO_NONE	0
MsgType	MQMT_DATAGRAM	8
Expiry	MQEI_UNLIMITED	-1
Feedback	MQFB_NONE	0
Encoding	MQENC_NATIVE	Depends on environment

Table 39. Initial values of fields in MQMD (continued)

Field name	Name of constant	Value of constant
CodedCharSetId	MQCCSI_Q_MGR	0
Format	MQFMT_NONE	Blanks
Priority	MQPRI_PRIORITY_AS_Q_DEF	-1
Persistence	MQPER_PERSISTENCE_AS_Q_DEF	2
MsgId	MQMI_NONE	Nulls
CorrelId	MQCI_NONE	Nulls
BackoutCount	None	0
ReplyToQ	None	Null string or blanks
ReplyToQMgr	None	Null string or blanks
UserIdentifier	None	Null string or blanks
AccountingToken	MQACT_NONE	Nulls
ApplIdentityData	None	Null string or blanks
PutApplType	MQAT_NO_CONTEXT	0
PutApplName	None	Null string or blanks
PutDate	None	Null string or blanks
PutTime	None	Null string or blanks
ApplOriginData	None	Null string or blanks
GroupId	MQGI_NONE	Nulls
MsgSeqNumber	None	1
Offset	None	0
MsgFlags	MQMF_NONE	0
OriginalLength	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};
   ```

C declaration

```
typedef 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    PutApplType;       /* Type of application that put the
                                  message */
  MQCHAR28  PutApplName;       /* 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 */
} MQMD;
```

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.
**    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

```
dcl
 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 PutApplType      fixed bin(31), /* Type of application that put the
                                       message */
  3 PutApplName      char(28),      /* Name of application that put the
                                       message */
  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 data
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 message
MQMD_PUTAPPLNAME        DS   CL28     Name of application that put
*                                     the message
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_LENGTH             EQU  *-MQMD   Length of structure
                        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      'Report options'
  MsgType             As Long      'Message type'
  Expiry              As Long      'Expiry time'
  Feedback            As Long      'Feedback or reason code'
  Encoding            As Long      'Data encoding'
  CodedCharSetId      As Long      'Coded character set identifier'
  Format              As String*8  'Format name'
  Priority            As Long      'Message priority'
  Persistence         As Long      'Message persistence'
  MsgId               As String*24 'Message identifier'
  CorrelId            As String*24 'Correlation identifier'
  BackoutCount        As Long      'Backout counter'
  ReplyToQ            As String*48 'Name of reply-to queue'
  ReplyToQMgr         As String*48 'Name of reply queue manager'
  UserIdentifier      As String*12 'User identifier'
  AccountingToken     As String*32 '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 String*24 'Group identifier'
  MsgSeqNumber        As Long      'Sequence number within group'
  Offset              As Long      'Offset of data in phsyical message'
                                   'from start of logical message'
  MsgFlags            As Long      'Message flags'
  OriginalLength      As Long      'Length of original message'
End Type
```

MQMD - Language declarations

Chapter 10. MQMDE - Message descriptor extension

The following table summarizes the fields in the structure.

Table 40. Fields in MQMDE

Field	Description	Page
StrucId	Structure identifier	189
Version	Structure version number	190
StrucLength	Length of MQMDE structure	189
Encoding	Numeric encoding of data that follows MQMDE	188
CodedCharSetId	Character set identifier of data that follows MQMDE	188
Format	Format name of data that follows MQMDE	188
Flags	General flags	188
GroupId	Group identifier	189
MsgSeqNumber	Sequence number of logical message within group	189
Offset	Offset of data in physical message from start of logical message	189
MsgFlags	Message flags	189
OriginalLength	Length of original message	189

Overview

Availability: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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: Character data in MQMDE must be in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute. Numeric data in MQMDE 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 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.

Note: On OS/2 and Windows NT, applications compiled with Micro Focus COBOL use a value of MQENC_NATIVE that is different from the

© Copyright IBM Corp. 1994, 2000

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 42 on page 190.

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 41.

Table 41. Queue-manager action when MQMDE specified on MQPUT or MQPUT1. This table shows the action taken by the queue manager when the application specifies an MQMDE structure at the start of the application message data on the MQPUT or MQPUT1 call.

MQMD version	Values of version-2 fields	Values of corresponding fields in MQMDE	Action taken by queue manager
1	–	Valid	MQMDE is honored
1	–	Not valid	Call fails with an appropriate reason code
1	–	MQMDE is in the wrong character set or encoding, or is an unsupported version	MQMDE is treated as message data
2	Default	Valid	MQMDE is honored
2	Default	Not valid	Call fails with an appropriate reason code
2	Default	MQMDE is in the wrong character set or encoding, or is an unsupported version	MQMDE is treated as message data
2	Not default	Valid	MQMDE is treated as message data
2	Not default	Not valid	Call fails with an appropriate reason code
2	Not default	MQMDE is in the wrong character set or encoding, or is an unsupported version	MQMDE is treated as message data

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 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 41 on page 186 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 *MQSeries Intercommunication* manual for further information).

Note: On OS/2 and Windows NT, 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.

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.

This value is supported in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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 9. MQMD - Message descriptor" on page 125 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 9. MQMD - Message descriptor" on page 125 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 9. MQMD - Message descriptor" on page 125. The initial value of this field is MQGI_NONE.

MsgFlags (MQLONG)

Message flags.

See the *MsgFlags* field described in "Chapter 9. MQMD - Message descriptor" on page 125. 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 9. MQMD - Message descriptor" on page 125. 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 9. MQMD - Message descriptor" on page 125. The initial value of this field is 0.

OriginalLength (MQLONG)

Length of original message.

See the *OriginalLength* field described in "Chapter 9. MQMD - Message descriptor" on page 125. 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.

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 42. Initial values of fields in MQMDE

Field name	Name of constant	Value of constant
StrucId	MQMDE_STRUC_ID	'MDEb'
Version	MQMDE_VERSION_2	2
StrucLength	MQMDE_LENGTH_2	72
Encoding	MQENC_NATIVE	Depends on environment
CodedCharSetId	MQCCSI_UNDEFINED	0
Format	MQFMT_NONE	Blanks
Flags	MQMDEF_NONE	0
GroupId	MQGI_NONE	Nulls
MsgSeqNumber	None	1
Offset	None	0
MsgFlags	MQMF_NONE	0
OriginalLength	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 {
  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 */
  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 */
} MQMDE;
```

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

```
dcl
 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 */
```

System/390 assembler declaration

```
MQMDE                   DSECT
MQMDE_STRUCID           DS   CL4      Structure identifier
MQMDE_VERSION           DS   F        Structure version number
MQMDE_STRUCLENGTH       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  Length of structure
                        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      'Data encoding'
  CodedCharSetId      As Long      'Coded character set identifier'
  Format              As String*8  'Format name'
  Flags               As Long      'General flags'
  GroupId             As String*24 'Group identifier'
  MsgSeqNumber        As Long      'Sequence number within group'
  Offset              As Long      'Offset of data in phsyical message'
                                   'from start of logical message'
  MsgFlags            As Long      'Message flags'
  OriginalLength      As Long      'Length of original message'
End Type
```

Chapter 11. MQOD - Object descriptor

The following table summarizes the fields in the structure.

Table 43. Fields in MQOD

Field	Description	Page
StrucId	Structure identifier	203
Version	Structure version number	203
ObjectType	Object type	200
ObjectName	Object name	197
ObjectQMgrName	Object queue manager name	198
DynamicQName	Dynamic queue name	196
AlternateUserId	Alternate user identifier	195
Note: The remaining fields are ignored if *Version* is less than MQOD_VERSION_2.		
RecsPresent	Number of object records present	200
KnownDestCount	Number of local queues opened successfully	196
UnknownDestCount	Number of remote queues opened successfully	203
InvalidDestCount	Number of queues that failed to open	196
ObjectRecOffset	Offset of first object record from start of MQOD	199
ResponseRecOffset	Offset of first response record from start of MQOD	201
ObjectRecPtr	Address of first object record	199
ResponseRecPtr	Address of first response record	202
Note: The remaining fields are ignored if *Version* is less than MQOD_VERSION_3.		
AlternateSecurityId	Alternate security identifier	194
ResolvedQName	Resolved queue name	201
ResolvedQMgrName	Resolved queue manager name	201

Overview

Availability:
- Version 1: All
- Versions 2 and 3: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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.

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

© Copyright IBM Corp. 1994, 2000

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: Character data in MQOD must be in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute. Numeric data in MQOD must be in the native machine encoding; this is given by MQENC_NATIVE.

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 NT, *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 OS/390, 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 NT 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 OS/390, 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.
- 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 OS/390, 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 OS/390, 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 OS/390:
  - 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 AS/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.
- 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 OS/390.
- 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.

  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.

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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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 OS/390, 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

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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

MQOD_VERSION_3
Version-3 object descriptor structure.

This version is supported in the following environments: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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:

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 44. Initial values of fields in MQOD

Field name	Name of constant	Value of constant
StrucId	MQOD_STRUC_ID	'ODbb'
Version	MQOD_VERSION_1	1
ObjectType	MQOT_Q	1
ObjectName	None	Null string or blanks
ObjectQMgrName	None	Null string or blanks
DynamicQName	None	'CSQ.*' on OS/390; 'AMQ.*' otherwise
AlternateUserId	None	Null string or blanks
RecsPresent	None	0
KnownDestCount	None	0
UnknownDestCount	None	0
InvalidDestCount	None	0
ObjectRecOffset	None	0
ResponseRecOffset	None	0
ObjectRecPtr	None	Null pointer or null bytes
ResponseRecPtr	None	Null pointer or null bytes
AlternateSecurityId	MQSID_NONE	Nulls
ResolvedQName	None	Null string or blanks
ResolvedQMgrName	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 {
  MQCHAR4   StrucId;             /* Structure identifier */
  MQLONG    Version;             /* Structure version number */
  MQLONG    ObjectType;          /* Object type */
  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 suc-
                                    cessfully */
  MQLONG    UnknownDestCount;    /* Number of remote queues opened suc-
                                    cessfully */
  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 */
} MQOD;
```

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-KNOWNDESTCOUNT      PIC S9(9) BINARY.
**    Number of remote queues opened successfully
   15 MQOD-UNKNOWNDESTCOUNT    PIC S9(9) BINARY.
**    Number of queues that failed to open
   15 MQOD-INVALIDDESTCOUNT    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-OBJECTRECPTR        POINTER.
**    Address of first response record
   15 MQOD-RESPONSERECPTR      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).
```

PL/I declaration

```
dcl
 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_KNOWNDESTCOUNT         DS   F        Number of local queues
*                                         opened successfully
MQOD_UNKNOWNDESTCOUNT       DS   F        Number of remote queues
*                                         opened successfully
MQOD_INVALIDDESTCOUNT       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_OBJECTRECPTR           DS   F        Address of first object
*                                         record
MQOD_RESPONSERECPTR         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   Length of structure
                            ORG  MQOD
MQOD_AREA                   DS   CL(MQOD_LENGTH)
```

TAL declaration

```
STRUCT       MQODˆDEF (*);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 String*32 'Address of first object record'
  ResponseRecPtr       As String*32 'Address of first response record'
  AlternateSecurityID  As String*40 'Alternate security identifier'
  ResolvedQName        As String*48 'Resolved queue name'
  ResolvedQMgrName     As String*48 'Resolved queue manager name'
End Type
```

Note: The *ObjectRecPtr* and *ResponseRecPtr* fields are not used, and are set to 32 null characters by default.

MQOD - Language declarations

Chapter 12. MQOR - Object record

The following table summarizes the fields in the structure.

Table 45. Fields in MQOR

Field	Description	Page
ObjectName	Object name	209
ObjectQMgrName	Object queue manager name	209

Overview

Availability: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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: Character data in MQOR must be in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute.

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.

This is the same as the *ObjectQMgrName* field in the MQOD structure (see MQOD for details).

© Copyright IBM Corp. 1994, 2000

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 46. Initial values of fields in MQOR

Field name	Name of constant	Value of constant
ObjectName	None	Null string or blanks
ObjectQMgrName	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 {
  MQCHAR48  ObjectName;      /* Object name */
  MQCHAR48  ObjectQMgrName;  /* Object queue manager name */
} MQOR;
```

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 13. MQPMO - Put message options

The following table summarizes the fields in the structure.

Table 47. Fields in MQPMO

Field	Description	Page
StrucId	Structure identifier	227
Version	Structure version number	227
Options	Options that control the action of MQPUT and MQPUT1	213
Timeout	Reserved	227
Context	Object handle of input queue	212
KnownDestCount	Number of messages sent successfully to local queues	212
UnknownDestCount	Number of messages sent successfully to remote queues	227
InvalidDestCount	Number of messages that could not be sent	212
ResolvedQName	Resolved name of destination queue	225
ResolvedQMgrName	Resolved name of destination queue manager	225
Note: The remaining fields are ignored if *Version* is less than MQPMO_VERSION_2.		
RecsPresent	Number of put message records or response records present	224
PutMsgRecFields	Flags indicating which MQPMR fields are present	222
PutMsgRecOffset	Offset of first put-message record from start of MQPMO	223
ResponseRecOffset	Offset of first response record from start of MQPMO	225
PutMsgRecPtr	Address of first put message record	224
ResponseRecPtr	Address of first response record	226

Overview

Availability:
- Version 1: All
- Version 2: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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 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.

© Copyright IBM Corp. 1994, 2000

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: Character data in MQPMO must be in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute. Numeric data in MQPMO must be in the native machine encoding; this is given by MQENC_NATIVE.

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 OS/390, Tandem 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 OS/390 Tandem 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_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, AS/400, Sun Solaris, Windows NT, plus MQSeries 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, AS/400, Sun Solaris, Windows NT, plus MQSeries 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”.

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 48 on page 217 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:

LOG ORD A "✓" means that the row applies only when the MQPMO_LOGICAL_ORDER option is specified.

MIG A "✓" means that the row applies only when the MQMF_MSG_IN_GROUP or MQMF_LAST_MSG_IN_GROUP option is specified.

SEG A "✓" means that the row applies only when the MQMF_SEGMENT or MQMF_LAST_SEGMENT option is specified.

A "(✓)" means that the row applies whether or not the MQMF_SEGMENT or MQMF_LAST_SEGMENT option is specified.

SEG OK A "✓" means that the row applies only when the MQMF_SEGMENTATION_ALLOWED option is specified.

A "(✓)" means that the row applies whether or not the MQMF_SEGMENTATION_ALLOWED option is specified.

Cur grp A "✓" means that the row applies only when a current message group exists prior to the call.

A "(✓)" means that the row applies whether or not a current message group exists prior to the call.

Cur log msg A "✓" means that the row applies only when a current logical message exists prior to the call.

A "(✓)" means that the row applies whether or not a current logical message exists prior to the call.

Other columns

These 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 48. 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*
✓						MQGI_NONE	1	0
✓			✓			New group id	1	0
✓		✓	(✓)			New group id	1	0
✓		✓	(✓)		✓	Previous group id	1	Previous offset + previous segment length
✓	✓	(✓)	(✓)			New group id	1	0
✓	✓	(✓)	(✓)	✓		Previous group id	Previous sequence number + 1	0
✓	✓	✓	(✓)	✓	✓	Previous group id	Previous sequence number	Previous offset + previous segment length
				(✓)	(✓)	MQGI_NONE	1	0
			✓	(✓)	(✓)	New group id if MQGI_NONE, else value in field	1	0
		✓	(✓)	(✓)	(✓)	New group id if MQGI_NONE, else value in field	1	Value in field
	✓		(✓)	(✓)	(✓)	New group id if MQGI_NONE, else value in field	Value in field	0
	✓	✓	(✓)	(✓)	(✓)	New group id if MQGI_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 MQMI_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.

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:

- Each successful MQPUT call that does *not* specify MQPMO_LOGICAL_ORDER 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, but the message or segment put is not the next one in the group or logical message. The call may however succeed with an MQCC_WARNING completion code. Table 49 on page 219 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 49. Outcome when MQPUT or MQCLOSE call not consistent with group and segment information

Current call	Previous call	
	MQPUT with MQPMO_LOGICAL_ORDER	MQPUT without MQPMO_LOGICAL_ORDER
MQPUT with MQPMO_LOGICAL_ORDER	MQCC_FAILED	MQCC_FAILED
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, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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.

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
UserIdentifier	Determined from the environment if possible; set to blanks otherwise.
AccountingToken	Determined from the environment if possible; set to MQACT_NONE otherwise.
ApplIdentityData	Set to blanks.
PutApplType	Determined from the environment.
PutApplName	Determined from the environment if possible; set to blanks otherwise.
PutDate	Set to date when message is put.
PutTime	Set to time when message is put.
ApplOriginData	Set to blanks.

For more information on message context, see the *MQSeries 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 *MQSeries 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 *MQSeries 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 *MQSeries 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 *MQSeries 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.

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 OS/390, 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

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 11. MQOD - Object descriptor" on page 193.

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

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 11. MQOD - Object descriptor" on page 193. 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, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

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 50. Initial values of fields in MQPMO

Field name	Name of constant	Value of constant
StrucId	MQPMO_STRUC_ID	'PMOb'
Version	MQPMO_VERSION_1	1
Options	MQPMO_NONE	0
Timeout	None	-1
Context	None	0
KnownDestCount	None	0
UnknownDestCount	None	0
InvalidDestCount	None	0
ResolvedQName	None	Null string or blanks
ResolvedQMgrName	None	Null string or blanks
RecsPresent	None	0
PutMsgRecFields	MQPMRF_NONE	0
PutMsgRecOffset	None	0
ResponseRecOffset	None	0
PutMsgRecPtr	None	Null pointer or null bytes
ResponseRecPtr	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 {
  MQCHAR4   StrucId;           /* Structure identifier */
  MQLONG    Version;           /* Structure version number */
  MQLONG    Options;           /* Options that control the action of
                                  MQPUT and MQPUT1 */
  MQLONG    Timeout;           /* Reserved */
  MQHOBJ    Context;           /* Object handle of input queue */
  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 */
} MQPMO;
```

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-KNOWNDESTCOUNT    PIC S9(9) BINARY.
**    Number of messages sent successfully to remote queues
   15 MQPMO-UNKNOWNDESTCOUNT  PIC S9(9) BINARY.
**    Number of messages that could not be sent
   15 MQPMO-INVALIDDESTCOUNT  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-PUTMSGRECPTR      POINTER.
**    Address of first response record
   15 MQPMO-RESPONSERECPTR    POINTER.
```

PL/I declaration

```
dcl
 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 */
  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 success-
                                        fully to local queues */
  3 UnknownDestCount  fixed bin(31), /* Number of messages sent success-
```

```
                                        fully 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_KNOWNDESTCOUNT        DS   F        Number of messages sent
*                                         successfully to local queues
MQPMO_UNKNOWNDESTCOUNT      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  Length of structure
                            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)      KNOWNDESTCOUNT;
INT(32)      UNKNOWNDESTCOUNT;
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 or MQPUT1'
  Timeout           As Long      'Reserved'
  Context           As Long      'Object handle of input queue'
  KnownDestCount    As Long      'Reserved'
  UnknownDestCount  As Long      'Reserved'
  InvalidDestCount  As Long      'Reserved'
  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 String*32 'Address of first put message record'
  ResponseRecPtr    As String*32 'Address of first response record'
End Type
```

MQPMO - Language declarations

Chapter 14. MQPMR - Put-message record

The following table summarizes the fields in the structure.

Table 51. Fields in MQPMR

Field	Description	Page
MsgId	Message identifier	235
CorrelId	Correlation identifier	234
GroupId	Group identifier	234
Feedback	Feedback or reason code	234
AccountingToken	Accounting token	234

Overview

Availability: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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: Numeric data in MQPMR must be in the native machine encoding; this is given by MQENC_NATIVE.

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**:

© Copyright IBM Corp. 1994, 2000

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 9. MQMD - Message descriptor" on page 125 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 above).

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 48 on page 217, 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 below).

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 {
  MQBYTE24  MsgId;            /* Message identifier */
  MQBYTE24  CorrelId;         /* Correlation identifier */
  MQBYTE24  GroupId;          /* Group identifier */
  MQLONG    Feedback;         /* Feedback or reason code */
  MQBYTE32  AccountingToken;  /* Accounting token */
} MQPMR;
```

COBOL declaration

```
**   MQPMR structure
  10 MQPMR.
**    Message identifier
   15 MQPMR-MSGID           PIC X(24).
**    Correlation identifier
   15 MQPMR-CORRELID        PIC X(24).
```

MQPMR - Language declarations

```
**    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

```
dcl
 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 String*24 'Message identifier'
  CorrelId          As String*24 'Correlation identifier'
  Feedback          As Long      'Feedback or reason code'
  AccountingToken   As String*32 'Accounting token'
End Type
```

Chapter 15. MQRFH - Rules and formatting header

The following table summarizes the fields in the structure.

Table 52. Fields in MQRFH

Field	Description	Page
StrucId	Structure identifier	239
Version	Structure version number	240
StrucLength	Total length of MQRFH including string containing name/value pairs	239
Encoding	Numeric encoding of data that follows *NameValueString*	238
CodedCharSetId	Character set identifier of data that follows *NameValueString*	237
Format	Format name of data that follows *NameValueString*	238
Flags	Flags	238
NameValueString	String containing name/value pairs	238

Overview

Availability: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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 a single-byte character set.

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.

© Copyright IBM Corp. 1994, 2000

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.

This value is supported in the following environments: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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.

The initial value of this field is MQRFH_STRUC_ID.

StrucLength (MQLONG)

Total length of MQRFH including string containing name/value pairs.

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 53. Initial values of fields in MQRFH

Field name	Name of constant	Value of constant
StrucId	MQRFH_STRUC_ID	'RFHb'
Version	MQRFH_VERSION_1	1
StrucLength	MQRFH_STRUC_LENGTH_FIXED	32
Encoding	MQENC_NATIVE	Depends on environment
CodedCharSetId	MQCCSI_UNDEFINED	0
Format	MQFMT_NONE	Blanks
Flags	MQRFH_NONE	0

Notes:

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

```
MQRFH MyRFH = {MQRFH_DEFAULT};
```

C declaration

```
typedef struct tagMQRFH {
  MQCHAR4  StrucId;         /* Structure identifier */
  MQLONG   Version;         /* Structure version number */
  MQLONG   StrucLength;     /* Total length of MQRFH including string
                               containing name/value pairs */
```

```
  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 */
} MQRFH;
```

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 string containing name/value
**    pairs
   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
                                     string containing name/value
                                     pairs */
  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 string containing
*                                       name/value pairs
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_LENGTH              EQU  *-MQRFH  Length of structure
                          ORG  MQRFH
MQRFH_AREA                DS   CL(MQRFH_LENGTH)
```

MQRFH - Language declarations

Chapter 16. MQRFH2 - Rules and formatting header version 2

The following table summarizes the fields in the structure.

Table 54. Fields in MQRFH2

Field	Description	Page
StrucId	Structure identifier	248
Version	Structure version number	248
StrucLength	Total length of MQRFH2 including all *NameValueLength* and *NameValueData* fields	248
Encoding	Numeric encoding of data that follows *NameValueData*	244
CodedCharSetId	Character set identifier of data that follows *NameValueData*	244
Format	Format name of data that follows *NameValueData*	244
Flags	Flags	244
NameValueCCSID	Character set identifier of *NameValueData*	245
NameValueLength	Length of *NameValueData*	247
NameValueData	Name/value data	245

Overview

Availability: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Purpose: 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.
- *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.

© Copyright IBM Corp. 1994, 2000

Fields

The MQRFH2 structure contains the following fields; the fields are described in **alphabetic order**:

CodedCharSetId (MQLONG)

Character set identifier of data that follows *NameValueData*.

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.

This value is supported in the following environments: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

The initial value of this field is MQCCSI_INHERIT.

Encoding (MQLONG)

Numeric encoding of data that follows *NameValueData*.

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 *NameValueData*.

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	Description
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 MQGMO_CONVERT option is specified on the MQGET call that retrieves the message, 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.

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.

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.

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:

- 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:

- If the value contains any of the following characters, each occurrence of the character must be replaced by the corresponding escape sequence:

Character	Escape sequence
&	&
<	<

- The following escape sequences are also defined, but their use is optional:

Character	Escape sequence
>	>
"	"
'	'

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:

```
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 55. Initial values of fields in MQRFH2

Field name	Name of constant	Value of constant
StrucId	MQRFH_STRUC_ID	'RFHb'
Version	MQRFH_VERSION_2	2
StrucLength	MQRFH_STRUC_LENGTH_FIXED_2	36
Encoding	MQENC_NATIVE	Depends on environment
CodedCharSetId	MQCCSI_INHERIT	-2
Format	MQFMT_NONE	Blanks
Flags	MQRFH_NONE	0
NameValueCCSID	None	1208

Notes:

1. The symbol 'b' represents a single blank character.
2. In the C programming language, the macro variable MQRFH2_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQRFH2 MyRFH2 = {MQRFH2_DEFAULT};
```

C declaration

```
typedef struct tagMQRFH2 {
  MQCHAR4  StrucId;         /* Structure identifier */
  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
                               NameValueData */
  MQLONG   CodedCharSetId;  /* Character set identifier of data that
                               follows NameValueData */
  MQCHAR8  Format;          /* Format name of data that follows
                               NameValueData */
  MQLONG   Flags;           /* Flags */
  MQLONG   NameValueCCSID;  /* Character set identifier of
                               NameValueData */
} MQRFH2;
```

COBOL declaration

```
**   MQRFH2 structure
  10 MQRFH.
**    Structure identifier
   15 MQRFH-STRUCID        PIC X(4).
**    Structure version number
   15 MQRFH-VERSION        PIC S9(9) BINARY.
**    Total length of MQRFH2 including all NameValueLength and
**    NameValueData fields
```

```
| 15 MQRFH-STRUCLENGTH    PIC S9(9) BINARY.
| **    Numeric encoding of data that follows NameValueData
| 15 MQRFH-ENCODING       PIC S9(9) BINARY.
| **    Character set identifier of data that follows NameValueData
| 15 MQRFH-CODEDCHARSETID PIC S9(9) BINARY.
| **    Format name of data that follows NameValueData
| 15 MQRFH-FORMAT         PIC X(8).
| **    Flags
| 15 MQRFH-FLAGS          PIC S9(9) BINARY.
| **    Character set identifier of NameValueData
| 15 MQRFH-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 NameValueData */
|   3 CodedCharSetId fixed bin(31), /* Character set identifier of data
|                                      that follows NameValueData */
|   3 Format         char(8),       /* Format name of data that follows
|                                      NameValueData */
|   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_STRUCLENGTH         DS   F        Total length of MQRFH2
| *                                       including all
| *                                       NameValueLength and
| *                                       NameValueData fields
| MQRFH_ENCODING            DS   F        Numeric encoding of data
| *                                       that follows NameValueData
| MQRFH_CODEDCHARSETID      DS   F        Character set identifier of
| *                                       data that follows
| *                                       NameValueData
| MQRFH_FORMAT              DS   CL8      Format name of data that
| *                                       follows NameValueData
| MQRFH_FLAGS               DS   F        Flags
| MQRFH_NAMEVALUECCSID      DS   F        Character set identifier of
| *                                       NameValueData
| MQRFH_LENGTH              EQU  *-MQRFH  Length of structure
|                           ORG  MQRFH
| MQRFH_AREA                DS   CL(MQRFH_LENGTH)
```

Chapter 17. MQRMH - Reference message header

The following table summarizes the fields in the structure.

Table 56. Fields in MQRMH

Field	Description	Page
StrucId	Structure identifier	257
Version	Structure version number	258
StrucLength	Total length of MQRMH, including strings at end of fixed fields, but not the bulk data	257
Encoding	Numeric encoding of bulk data	255
CodedCharSetId	Character set identifier of bulk data	252
Format	Format name of bulk data	255
Flags	Reference message flags	255
ObjectType	Object type	256
ObjectInstanceId	Object instance identifier	256
SrcEnvLength	Length of source environment data	256
SrcEnvOffset	Offset of source environment data	256
SrcNameLength	Length of source object name	257
SrcNameOffset	Offset of source object name	257
DestEnvLength	Length of destination environment data	254
DestEnvOffset	Offset of destination environment data	254
DestNameLength	Length of destination object name	254
DestNameOffset	Offset of destination object name	254
DataLogicalLength	Length of bulk data	253
DataLogicalOffset	Low offset of bulk data	253
DataLogicalOffset2	High offset of bulk data	254

Overview

Availability: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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.

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

© Copyright IBM Corp. 1994, 2000

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 MQXQH 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* 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.

This value is supported in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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*.

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.

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.

ObjectInstanceId (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.

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 57. Initial values of fields in MQRMH

Field name	Name of constant	Value of constant
StrucId	MQRMH_STRUC_ID	'RMHb'
Version	MQRMH_VERSION_1	1
StrucLength	None	0
Encoding	MQENC_NATIVE	Depends on environment
CodedCharSetId	MQCCSI_UNDEFINED	0
Format	MQFMT_NONE	Blanks
Flags	MQRMHF_NOT_LAST	0
ObjectType	None	Blanks
ObjectInstanceId	MQOII_NONE	Nulls
SrcEnvLength	None	0
SrcEnvOffset	None	0
SrcNameLength	None	0
SrcNameOffset	None	0
DestEnvLength	None	0
DestEnvOffset	None	0
DestNameLength	None	0
DestNameOffset	None	0
DataLogicalLength	None	0
DataLogicalOffset	None	0
DataLogicalOffset2	None	0

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 {
  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 */
} MQRMH;
```

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).
**    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-DATALOGICALLENGTH  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

```
dcl
 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 environ-
                                         ment data */
  3 DestEnvOffset      fixed bin(31), /* Offset of destination environ-
                                         ment 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_STRUCLENGTH       DS   F        Total length of MQRMH,
*                                     including 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_DATALOGICALLENGTH       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  Length of structure
                              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      'Data encoding'
  CodedCharSetId      As Long      'Coded character set identifier'
  Format              As String*8  'Format name'
  Flags               As Long      'Reference message flags'
  ObjectType          As String*8  'Object type'
  ObjectInstanceId    As String*24 'Object instance identifier'
  SrcEnvLength        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'
  DestNameLenght      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 18. MQRR - Response record

The following table summarizes the fields in the structure.

Table 58. Fields in MQRR

Field	Description	Page
CompCode	Completion code for queue	263
Reason	Reason code for queue	263

Overview

Availability: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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: Numeric data in MQRR must be in the native machine encoding; this is given by MQENC_NATIVE.

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.

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.

© Copyright IBM Corp. 1994, 2000

This is always an output field. The initial value of this field is MQRC_NONE.

Initial values and language declarations

Table 59. 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 {
  MQLONG  CompCode;  /* Completion code for queue */
  MQLONG  Reason;    /* Reason code for queue */
} MQRR;
```

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 19. MQTM - Trigger message

The following table summarizes the fields in the structure.

Table 60. Fields in MQTM

Field	Description	Page
StrucId	Structure identifier	269
Version	Structure version number	270
QName	Name of triggered queue	268
ProcessName	Name of process object	268
TriggerData	Trigger data	269
ApplType	Application type	267
ApplId	Application identifier	267
EnvData	Environment data	268
UserData	User data	269

Overview

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

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 MQSeries Trigger Monitor Interface (TMI), which is one of the MQSeries 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 20. MQTMC2 - Trigger message 2 (character format)" on page 273.

- On OS/390, 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.

© Copyright IBM Corp. 1994, 2000

- On AS/400, the trigger-monitor application provided with MQSeries 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 *MQSeries 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
StrucId	MQMD_STRUC_ID
Version	MQMD_VERSION_1
Report	MQRO_NONE
MsgType	MQMT_DATAGRAM
Expiry	MQEI_UNLIMITED
Feedback	MQFB_NONE
Encoding	MQENC_NATIVE
CodedCharSetId	Queue manager's *CodedCharSetId* attribute
Format	MQFMT_TRIGGER
Priority	Initiation queue's *DefPriority* attribute
Persistence	MQPER_NOT_PERSISTENT
MsgId	A unique value
CorrelId	MQCI_NONE
BackoutCount	0
ReplyToQ	Blanks
ReplyToQMgr	Name of queue manager
UserIdentifier	Blanks
AccountingToken	MQACT_NONE
ApplIdentityData	Blanks
PutApplType	MQAT_QMGR, or as appropriate for the message channel agent
PutApplName	First 28 bytes of the queue-manager name
PutDate	Date when trigger message is sent
PutTime	Time when trigger message is sent
ApplOriginData	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 it 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**:

ApplId (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 *ApplId* attribute of the process object identified by the *ProcessName* field; see "Chapter 41. Attributes for process definitions" on page 463 for details of this attribute. The content of this data is of no significance to the queue manager.

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 OS/390, *ApplId* 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 DOS client, OS/2, and Windows systems, the program name may be prefixed with a drive and directory path.
- On AS/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, *ApplId* 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.

ApplType (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 *ApplType* attribute of the process object identified by the *ProcessName* field; see "Chapter 41. Attributes for process definitions" on page 463 for details of this attribute. The content of this data is of no significance to the queue manager.

ApplType 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.
MQAT_DOS	DOS client application.
MQAT_IMS	IMS application.
MQAT_MVS	OS/390 or TSO application (same value as MQAT_OS390).

MQAT_NOTES_AGENT	Lotus Notes Agent application.
MQAT_NSK	Tandem NonStop Kernel application.
MQAT_OS2	OS/2 or Presentation Manager application.
MQAT_OS390	OS/390 application.
MQAT_OS400	AS/400 application.
MQAT_UNIX	UNIX application.
MQAT_VMS	Digital OpenVMS application.
MQAT_WINDOWS	Windows client, Windows 3.1 application.
MQAT_WINDOWS_NT	Windows NT, Windows 95, Windows 98 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 41. Attributes for process definitions" on page 463 for details of this attribute. The content of this data is of no significance to the queue manager.

On OS/390, 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 39. Attributes for queues" on page 427 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 39. Attributes for queues" on page 427 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 39. Attributes for queues" on page 427 for details of this attribute. The content of this data is of no significance to the queue manager.

- On OS/390, 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.
  - 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 41. Attributes for process definitions" on page 463 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 61. Initial values of fields in MQTM

Field name	Name of constant	Value of constant
StrucId	MQTM_STRUC_ID	'TMbb'
Version	MQTM_VERSION_1	1
QName	None	Null string or blanks
ProcessName	None	Null string or blanks
TriggerData	None	Null string or blanks
ApplType	None	0
ApplId	None	Null string or blanks
EnvData	None	Null string or blanks
UserData	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 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 {
  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 */
} MQTM;
```

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 */
  3 ApplId      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   Length of structure
                            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;
```

MQTM - Language declarations

```
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 20. MQTMC2 - Trigger message 2 (character format)

The following table summarizes the fields in the structure.

Table 62. Fields in MQTMC2

Field	Description	Page
StrucId	Structure identifier	274
Version	Structure version number	275
QName	Name of triggered queue	274
ProcessName	Name of process object	274
TriggerData	Trigger data	274
ApplType	Application type	274
ApplId	Application identifier	274
EnvData	Environment data	274
UserData	User data	275
QMgrName	Queue manager name	274

Overview

Availability: Not VSE/ESA, Windows 3.1, Windows 95, Windows 98.

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 MQSeries Trigger Monitor Interface (TMI), which is one of the MQSeries 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 OS/390, for an MQAT_IMS application that is started using the CSQQTRMN application, an MQTMC2 structure is made available to the started application.
- On AS/400, the trigger monitor application provided with MQSeries 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.

© Copyright IBM Corp. 1994, 2000

Fields

The MQTMC2 structure contains the following fields; the fields are described in **alphabetic order**:

AppId (MQCHAR256)

Application identifier.

See the *ApplId* field in the MQTM structure.

ApplType (MQCHAR4)

Application type.

This field always contains blanks, whatever the value in the *ApplType* 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 63. Initial values of fields in MQTMC2

Field name	Name of constant	Value of constant
StrucId	MQTMC_STRUC_ID	'TMCb'
Version	MQTMC_VERSION_2	'bbb2'
QName	None	Null string or blanks
ProcessName	None	Null string or blanks
TriggerData	None	Null string or blanks
ApplType	None	Blanks
ApplId	None	Null string or blanks
EnvData	None	Null string or blanks
UserData	None	Null string or blanks
QMgrName	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 {
  MQCHAR4    StrucId;      /* Structure identifier */
  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 */
} MQTMC2;
```

COBOL declaration

```
**   MQTMC2 structure
  10 MQTMC.
**    Structure identifier
   15 MQTMC-STRUCID     PIC X(4).
**    Structure version number
   15 MQTMC-VERSION     PIC X(4).
**    Name of triggered queue
   15 MQTMC-QNAME       PIC X(48).
**    Name of process object
   15 MQTMC-PROCESSNAME PIC X(48).
**    Trigger data
   15 MQTMC-TRIGGERDATA PIC X(64).
**    Application type
   15 MQTMC-APPLTYPE    PIC X(4).
**    Application identifier
   15 MQTMC-APPLID      PIC X(256).
**    Environment data
   15 MQTMC-ENVDATA     PIC X(128).
**    User data
   15 MQTMC-USERDATA    PIC X(128).
**    Queue manager name
   15 MQTMC-QMGRNAME    PIC X(48).
```

PL/I declaration

```
dcl
 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  Length of structure
                         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 21. MQWIH - Work information header

The following table summarizes the fields in the structure.

Table 64. Fields in MQWIH

Field	Description	Page
StrucId	Structure identifier	281
Version	Structure version number	282
StrucLength	Length of MQWIH structure	281
Encoding	Numeric encoding of data that follows MQWIH	280
CodedCharSetId	Character-set identifier of data that follows MQWIH	279
Format	Format name of data that follows MQWIH	280
Flags	Flags	280
ServiceName	Service name	281
ServiceStep	Service step name	281
MsgToken	Message token	281
Reserved	Reserved	281

Overview

Availability: AIX, HP-UX, OS/390, OS/2, Sun Solaris, Windows NT, plus MQSeries 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 OS/390 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 OS/390 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.

© Copyright IBM Corp. 1994, 2000

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.

This value is supported in the following environments: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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.

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 65. Initial values of fields in MQWIH

Field name	Name of constant	Value of constant
StrucId	MQWIH_STRUC_ID	'WIHb'
Version	MQWIH_VERSION_1	1
StrucLength	MQWIH_LENGTH_1	120
Encoding	None	0
CodedCharSetId	MQCCSI_UNDEFINED	0
Format	MQFMT_NONE	Blanks
Flags	MQWIH_NONE	0
ServiceName	None	Blanks
ServiceStep	None	Blanks
MsgToken	MQMTOK_NONE	Nulls
Reserved	None	Blanks

Notes:

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:

   ```
   MQWIH MyWIH = {MQWIH_DEFAULT};
   ```

C declaration

```
typedef struct tagMQWIH {
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQLONG    StrucLength;     /* 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 */
} MQWIH;
```

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

```
dcl
 1 MQWIH based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 StrucLength    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
MQWIH_CODEDCHARSETID     DS   F        Character-set identifier of
*                                      data that follows MQWIH
```

MQWIH - Language declarations

```
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  Length of structure
                            ORG  MQWIH
MQWIH_AREA                  DS   CL(MQWIH_LENGTH)
```

Chapter 22. MQXP - Exit parameter block (OS/390 only)

The following table summarizes the fields in the structure.

Table 66. Fields in MQXP

Field	Description	Page
StrucId	Structure identifier	288
Version	Structure version number	288
ExitId	Exit identifier	286
ExitReason	Reason for invocation of exit	286
ExitResponse	Response from exit	287
ExitCommand	API call code	285
ExitParmCount	Parameter count	286
ExitUserArea	User area	287

Overview

Availability: OS/390.

Purpose: The MQXP structure is used as an input/output parameter to the API-crossing exit. For more information on this exit, see the *MQSeries 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.

© Copyright IBM Corp. 1994, 2000

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.

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
Continue normally.

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_OK, 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.

StrucId (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 {
  MQCHAR4   StrucId;        /* 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 */
} MQXP;
```

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   Length of structure
                        ORG  MQXP
MQXP_AREA               DS   CL(MQXP_LENGTH)
```

MQXP - Language declarations

Chapter 23. MQXQH - Transmission queue header

The following table summarizes the fields in the structure.

Table 67. Fields in MQXQH

Field	Description	Page
StrucId	Structure identifier	295
Version	Structure version number	295
RemoteQName	Name of destination queue	294
RemoteQMgrName	Name of destination queue manager	294
MsgDesc	Original message descriptor	294

Overview

Availability: All.

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: Character data in MQXQH must be in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute. Numeric data in MQXQH 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 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.
- 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).

© Copyright IBM Corp. 1994, 2000

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
StrucId	MQMD_STRUC_ID
Version	MQMD_VERSION_2
Report	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.)
MsgType	Copied from the embedded message descriptor.
Expiry	Copied from the embedded message descriptor.
Feedback	Copied from the embedded message descriptor.
Encoding	MQENC_NATIVE (see note below)
CodedCharSetId	Queue manager's *CodedCharSetId* attribute.
Format	MQFMT_XMIT_Q_HEADER
Priority	Copied from the embedded message descriptor.
Persistence	Copied from the embedded message descriptor.
MsgId	A new value is generated by the queue manager. This message identifier is different from the *MsgId* that the queue manager may have generated for the embedded message descriptor (see above).
CorrelId	The *MsgId* from the embedded message descriptor.
BackoutCount	0
ReplyToQ	Copied from the embedded message descriptor.
ReplyToQMgr	Copied from the embedded message descriptor.
UserIdentifier	Copied from the embedded message descriptor.
AccountingToken	Copied from the embedded message descriptor.
ApplIdentityData	Copied from the embedded message descriptor.
PutApplType	MQAT_QMGR
PutApplName	First 28 bytes of the queue-manager name.
PutDate	Date when message was put on transmission queue.
PutTime	Time when message was put on transmission queue.
ApplOriginData	Blanks
GroupId	MQGI_NONE
MsgSeqNumber	1
Offset	0
MsgFlags	MQMF_NONE
OriginalLength	MQOL_UNDEFINED

- On OS/2 and Windows NT, 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	*PutApplType*
ApplIdentityData	*PutDate*
ApplOriginData	*PutTime*
PutApplName	*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, 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 68. Initial values of fields in MQXQH

Field name	Name of constant	Value of constant
StrucId	MQXQH_STRUC_ID	'XQHb'
Version	MQXQH_VERSION_1	1
RemoteQName	None	Null string or blanks
RemoteQMgrName	None	Null string or blanks
MsgDesc	Same names and values as MQMD; see Table 39 on page 178	

Table 68. 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 {
  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 */
} MQXQH;
```

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).
**     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

```
dcl
 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 */
```

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  Length of structure
                               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'
```

```
  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 24. Call descriptions 305
Conventions used in the call descriptions 305
Using the calls in the C language. 307
Declaring the Buffer parameter 307

Chapter 25. MQBACK - Back out changes . . . 309
Syntax. 309
Parameters 309
Hconn (MQHCONN) – input 309
CompCode (MQLONG) – output. 309
Reason (MQLONG) – output 309
Usage notes 310
Language invocations 312
C invocation. 312
COBOL invocation 312
PL/I invocation (AIX, OS/2, OS/390, Windows NT) 313
System/390 assembler invocation (OS/390 only) 313
TAL invocation (Tandem NSK only) 313
Visual Basic invocation (Windows only) . . . 313

Chapter 26. MQBEGIN - Begin unit of work . . 315
Syntax. 315
Parameters 315
Hconn (MQHCONN) – input 315
BeginOptions (MQBO) – input/output 315
CompCode (MQLONG) – output. 315
Reason (MQLONG) – output 315
Usage notes 316
Language invocations 317
C invocation. 318
COBOL invocation 318
PL/I invocation (AIX, OS/2, Windows NT) . . 318
Visual Basic invocation (Windows only) . . . 318

Chapter 27. MQCLOSE - Close object 319
Syntax. 319
Parameters 319
Hconn (MQHCONN) – input 319
Hobj (MQHOBJ) – input/output 319
Options (MQLONG) – input 319
CompCode (MQLONG) – output. 321
Reason (MQLONG) – output 321
Usage notes 322
Language invocations 324
C invocation. 324
COBOL invocation 324
PL/I invocation (AIX, OS/2, OS/390, VSE/ESA, Windows NT 325
System/390 assembler invocation (OS/390 only) 325
TAL invocation (Tandem NSK only) 325
Visual Basic invocation (Windows only) . . . 325

Chapter 28. MQCMIT - Commit changes . . . 327
Syntax. 327
Parameters 327
Hconn (MQHCONN) – input 327
CompCode (MQLONG) – output. 327
Reason (MQLONG) – output 327
Usage notes 328
Language invocations 330
C invocation. 330
COBOL invocation 330
PL/I invocation (AIX, OS/2, OS/390, Windows NT) 330
System/390 assembler invocation (OS/390 only) 331
TAL invocation (Tandem NSK only) 331
Visual Basic invocation (Windows only) . . . 331

Chapter 29. MQCONN - Connect queue manager 333
Syntax. 333
Parameters 333
QMgrName (MQCHAR48) – input 333
Hconn (MQHCONN) – output 335
CompCode (MQLONG) – output. 336
Reason (MQLONG) – output 336
Usage notes 338
Language invocations 339
C invocation. 339
COBOL invocation 340
PL/I invocation (AIX, OS/2, OS/390, VSE/ESA, Windows NT) 340
System/390 assembler invocation (OS/390 only) 340
TAL invocation (Tandem NSK only) 340
Visual Basic invocation (Windows only) . . . 340

Chapter 30. MQCONNX - Connect queue manager (extended) 341
Syntax. 341
Parameters 341
QMgrName (MQCHAR48) – input 341
ConnectOpts (MQCNO) – input/output . . . 341
Hconn (MQHCONN) – output 341
CompCode (MQLONG) – output. 341
Reason (MQLONG) – output 341
Language invocations 342
C invocation. 342
COBOL invocation 342
PL/I invocation (AIX, OS/2, OS/390, Windows NT) 342
System/390 assembler invocation (OS/390 only) 343
Visual Basic invocation (Windows only) . . . 343

Chapter 31. MQDISC - Disconnect queue manager 345
Syntax. 345
Parameters 345
Hconn (MQHCONN) – input/output 345
CompCode (MQLONG) – output. 345
Reason (MQLONG) – output 346

© Copyright IBM Corp. 1994, 2000

Usage notes 347
Language invocations 348
C invocation. 348
COBOL invocation 348
PL/I invocation (AIX, OS/2, OS/390, VSE/ESA, Windows NT) 348
System/390 assembler invocation (OS/390 only) 348
TAL invocation (Tandem NSK only) 348
Visual Basic invocation (Windows only) . . . 348

Chapter 32. MQGET - Get message 349
Syntax. 349
Parameters 349
Hconn (MQHCONN) – input 349
Hobj (MQHOBJ) – input. 349
MsgDesc (MQMD) – input/output 349
GetMsgOpts (MQGMO) – input/output . . . 350
BufferLength (MQLONG) – input 350
Buffer (MQBYTE×BufferLength) – output . . . 350
DataLength (MQLONG) – output 351
CompCode (MQLONG) – output. 351
Reason (MQLONG) – output 351
Usage notes 355
Language invocations 359
C invocation. 359
COBOL invocation 359
PL/I invocation (AIX, OS/2, OS/390, VSE/ESA, Windows NT) 360
System/390 assembler invocation (OS/390 only) 360
TAL invocation (Tandem NSK only) 360
Visual Basic invocation (Windows only) . . . 360

Chapter 33. MQINQ - Inquire about object attributes 363
Syntax. 363
Parameters 363
Hconn (MQHCONN) – input 363
Hobj (MQHOBJ) – input. 363
SelectorCount (MQLONG) – input 363
Selectors (MQLONG×SelectorCount) – input 364
IntAttrCount (MQLONG) – input 367
IntAttrs (MQLONG×IntAttrCount) – output . . 367
CharAttrLength (MQLONG) – input 368
CharAttrs (MQCHAR×CharAttrLength) – output. 368
CompCode (MQLONG) – output. 368
Reason (MQLONG) – output 369
Usage notes 370
Language invocations 371
C invocation. 371
COBOL invocation 371
PL/I invocation (AIX, OS/2, OS/390, VSE/ESA, Windows NT) 372
System/390 assembler invocation (OS/390 only) 372
TAL invocation (Tandem NSK only) 372
Visual Basic invocation (Windows only) . . . 373

Chapter 34. MQOPEN - Open object 375
Syntax. 375
Parameters 375
Hconn (MQHCONN) – input 375
ObjDesc (MQOD) – input/output 375
Options (MQLONG) – input 376
Hobj (MQHOBJ) – output 382
CompCode (MQLONG) – output. 382
Reason (MQLONG) – output 383
Usage notes 385
Language invocations 391
C invocation. 391
COBOL invocation 391
PL/I invocation (AIX, OS/2, OS/390, VSE/ESA, Windows NT) 391
System/390 assembler invocation (OS/390 only) 392
TAL invocation (Tandem NSK only) 392
Visual Basic invocation (Windows only) . . . 392

Chapter 35. MQPUT - Put message 393
Syntax. 393
Parameters 393
Hconn (MQHCONN) – input 393
Hobj (MQHOBJ) – input. 393
MsgDesc (MQMD) – input/output 393
PutMsgOpts (MQPMO) – input/output . . . 394
BufferLength (MQLONG) – input 394
Buffer (MQBYTE×BufferLength) – input . . . 395
CompCode (MQLONG) – output. 395
Reason (MQLONG) – output 395
Usage notes 399
Language invocations 403
C invocation. 403
COBOL invocation 403
PL/I invocation (AIX, OS/2, OS/390, VSE/ESA, Windows NT) 404
System/390 assembler invocation (OS/390 only) 404
TAL invocation (Tandem NSK only) 404
Visual Basic invocation (Windows only) . . . 404

Chapter 36. MQPUT1 - Put one message . . . 405
Syntax. 405
Parameters 405
Hconn (MQHCONN) – input 405
ObjDesc (MQOD) – input/output 405
MsgDesc (MQMD) – input/output 405
PutMsgOpts (MQPMO) – input/output . . . 406
BufferLength (MQLONG) – input 406
Buffer (MQBYTE×BufferLength) – input . . . 406
CompCode (MQLONG) – output. 406
Reason (MQLONG) – output 406
Usage notes 411
Language invocations 412
C invocation. 412
COBOL invocation 412
PL/I invocation (AIX, OS/2, OS/390, VSE/ESA, Windows NT) 413
System/390 assembler invocation (OS/390 only) 413
TAL invocation (Tandem NSK only) 413
Visual Basic invocation (Windows only) . . . 414

Chapter 37. MQSET - Set object attributes . . . 415
Syntax. 415
Parameters 415
Hconn (MQHCONN) – input 415

Hobj (MQHOBJ) – input 415
SelectorCount (MQLONG) – input 415
Selectors (MQLONG×SelectorCount) – input 415
IntAttrCount (MQLONG) – input 416
IntAttrs (MQLONG×IntAttrCount) – input . . 416
CharAttrLength (MQLONG) – input 417
CharAttrs (MQCHAR×CharAttrLength) – input 417
CompCode (MQLONG) – output 417
Reason (MQLONG) – output 417
Usage notes 419
Language invocations 419
C invocation. 420
COBOL invocation 420
PL/I invocation (AIX, OS/2, OS/390, VSE/ESA, Windows NT) 420
System/390 assembler invocation (OS/390 only) 421
TAL invocation (Tandem NSK only) 421
Visual Basic invocation (Windows only) . . . 421

Chapter 38. MQSYNC - Synchronize statistics updates (Tandem NSK only) 423
Syntax. 423
Parameters 423
Language invocations 424
C language invocation 424
COBOL language invocation 424
TAL language invocation 424

Chapter 24. 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 (Tandem 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 599.

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

© Copyright IBM Corp. 1994, 2000

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 489.

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. MQSeries constants" on page 547 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;
```

Chapter 25. MQBACK - Back out changes

The MQBACK call indicates to the queue manager that all of 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 OS/390, this call is used only by batch programs (including IMS batch DL/I programs).
- On AS/400, this call is not supported for applications running in compatibility mode.
- On Tandem NonStop Kernel, this call can be issued by the application but always returns completion code MQCC_FAILED and reason code MQRC_ENVIRONMENT_ERROR.
- 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:

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.

© Copyright IBM Corp. 1994, 2000

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.
MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call reentered 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 489 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 26. MQBEGIN - Begin unit of work" on page 315.
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 OS/390, 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 OS/390 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.
     - 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 AS/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 Tandem 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 31. MQDISC - Disconnect queue manager" on page 345 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:
   - 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 13. MQPMO - Put message options" on page 211, and the MQGMO_LOGICAL_ORDER option described in "Chapter 7. MQGMO - Get-message options" on page 81.

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 29. MQCONN - Connect queue manager" on page 333 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

This call is supported in the following programming languages:

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 (AIX, OS/2, OS/390, Windows NT)

```
call MQBACK (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 (OS/390 only)

```
        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 (Tandem NSK only)

```
INT(32)   .EXT Hconn;
INT(32)   .EXT CC;
INT(32)   .EXT Reason;

CALL  MQBACK(HConn, CC, Reason);
```

Visual Basic invocation (Windows only)

```
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 26. 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, AS/400, Sun Solaris, Windows NT.

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.

BeginOptions (MQBO) – input/output

Options that control the action of MQBEGIN.

See "Chapter 2. MQBO - Begin options" on page 29 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.

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:

© Copyright IBM Corp. 1994, 2000

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_BO_ERROR
(2134, X'856') Begin-options structure not valid.
MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call reentered 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 489.

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 31. MQDISC - Disconnect queue manager" on page 345 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 AS/400, the three types of unit of work are supported as follows:
   - **Queue-manager-coordinated local units 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 units of work** are not supported.
   - **Externally-coordinated global units 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 AS/400 `COMMIT` and `ROLLBACK` operations apply to MQ resources as well as to resources belonging to other participating resource managers.

Language invocations

This call is supported in the following programming languages:

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 (AIX, OS/2, Windows NT)

```
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 (Windows only)

```
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 27. 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 OS/390 for CICS applications, and on AS/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 OS/390, *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 39. Attributes for queues" on page 427). The close options are summarized in Table 69 on page 321.

© Copyright IBM Corp. 1994, 2000

MQCLOSE - Parameters

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 OS/390, 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 OS/390, 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.

Table 69. 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_LOAD_ERROR
(2183, X'887') Unable to load API crossing exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call reentered before previous call complete.

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 489 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 7. MQGMO - Get-message options" on page 81).
- On OS/390, 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 7. MQGMO - Get-message options" on page 81). 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 7. MQGMO - Get-message options" on page 81.

     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 OS/390, 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.
   - 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 AS/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.

Language invocations

This call is supported in the following programming languages:

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 (AIX, OS/2, OS/390, VSE/ESA, Windows NT

```
call MQCLOSE (Hconn, Hobj, Options, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn     fixed bin(31);  /* Connection handle */
dcl Hobj      fixed bin(31);  /* Object handle */
dcl Options   fixed bin(31);  /* Options that control the action of
                                 MQCLOSE */
dcl CompCode  fixed bin(31);  /* Completion code */
dcl Reason    fixed bin(31);  /* Reason code qualifying CompCode */
```

System/390 assembler invocation (OS/390 only)

```
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 (Tandem NSK only)

```
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 (Windows only)

```
MQCLOSE Hconn, Hobj, Options, CompCode, Reason
```

Declare the parameters as follows:

```
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 28. 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 OS/390, the call is used only by batch programs (including IMS batch DL/I programs).
- On AS/400, this call is not supported for applications running in compatibility mode.
- On Tandem NonStop Kernel, this call can be issued by the application but always returns completion code MQCC_FAILED and reason code MQRC_ENVIRONMENT_ERROR.
- 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.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
: (0, X'000') No reason to report.

© Copyright IBM Corp. 1994, 2000

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_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call reentered 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 489 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 26. MQBEGIN - Begin unit of work" on page 315.
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 OS/390, 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 SRRCMIT call provided by the OS/390 Recoverable Resource Service (RRS) should be used. The SRRCMIT call commits changes to resources belonging to the resource managers that have been enabled for RRS coordination.

  - 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 AS/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 Tandem 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 31. MQDISC - Disconnect queue manager" on page 345 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.

   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 13. MQPMO - Put

message options" on page 211, and the MQGMO_LOGICAL_ORDER option described in "Chapter 7. MQGMO - Get-message options" on page 81.

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 (DIGITAL) OpenVMS, OS/2, Tandem NonStop Kernel, UNIX systems, and Windows NT.

Language invocations

This call is supported in the following programming languages:

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 (AIX, OS/2, OS/390, Windows NT)

```
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 (OS/390 only)

```
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 (Tandem NSK only)

```
INT(32)   .EXT Hconn;
INT(32)   .EXT CC;
INT(32)   .EXT Reason;
```

Visual Basic invocation (Windows only)

```
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 29. 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 OS/390, 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 AS/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 AS/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 OS/390, 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 AS/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.

© Copyright IBM Corp. 1994, 2000

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 OS/390:
  - 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 338).
  - For OS/390 batch and TSO, only queue managers that reside on the same system as the application are connectable (see Usage note 6 on page 338).
- 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: 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. 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 OS/390. 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 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: OS/390, Windows 3.1, Windows 95, Windows 98. However, OS/390 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: OS/390, 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.

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.

The scope of the handle is restricted to the smallest unit of parallel processing supported by the platform on which the application is running; the handle is not valid outside the unit of parallel processing from which the MQCONN call was issued.

- On Compaq (DIGITAL) OpenVMS, the scope of the handle is the thread issuing the call.
- On PC DOS, the scope of the handle is the system.
- On OS/390, the scope of the handle is:
  - For CICS, the CICS task issuing the call
  - For IMS, the task issuing the call, up to the next syncpoint; this excludes any subtasks of the task
  - For OS/390 batch and TSO, the task issuing the call; this excludes any subtasks of the task
- On OS/2, the scope of the handle is the thread issuing the call.
- On AS/400, the scope of the handle is the job issuing the call.
- On Tandem NonStop Kernel, the scope of the handle is the process.
- On AIX, HP-UX, Sun Solaris, and other UNIX systems, the scope of the handle is the thread issuing the call.
- On VSE/ESA, the scope of the handle is the CICS transaction.
- On Windows 3.1, and for Windows 3.1 applications running on Windows 95, Windows 98, Windows NT, or Win-OS2, the scope of the handle is the process issuing the call.
- On Windows 95, Windows 98 and Windows NT, the scope of the handle is the thread issuing the call.

On OS/390 for CICS applications, and on AS/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.

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_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.
MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call reentered before previous call complete.
MQRC_CLUSTER_EXIT_LOAD_ERROR
(2267, X'8DB') Unable to load cluster workload exit.
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.
MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.
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_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_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 489.

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 OS/390:
   - 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 OS/390, 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.

- 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 *MQSeries for OS/390 System Setup Guide*.

7. On AS/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.

     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 AS/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

This call is supported in the following programming languages:

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 (AIX, OS/2, OS/390, VSE/ESA, Windows NT)

```
call MQCONN (QMgrName, Hconn, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl QMgrName  char(48);       /* Name of queue manager */
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 (OS/390 only)

```
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 (Tandem NSK only)

```
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 (Windows only)

```
MQCONN Name, Hconn, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Name     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'
```

Chapter 30. 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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.
- On AS/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 29. MQCONN - Connect queue manager" on page 333 for details.

ConnectOpts (MQCNO) – input/output

Options that control the action of MQCONNX.

See "Chapter 4. MQCNO - Connect options" on page 51 for details.

Hconn (MQHCONN) – output

Connection handle.

See the *Hconn* parameter described in "Chapter 29. MQCONN - Connect queue manager" on page 333 for details.

CompCode (MQLONG) – output

Completion code.

See the *CompCode* parameter described in "Chapter 29. MQCONN - Connect queue manager" on page 333 for details.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

© Copyright IBM Corp. 1994, 2000

See the *Reason* parameter described in "Chapter 29. MQCONN - Connect queue manager" on page 333 for details of possible reason codes.

The following additional reason codes can be returned by the MQCONNX call:

If *CompCode* is MQCC_FAILED:

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_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

For more information on these reason codes, see "Appendix A. Return codes" on page 489.

Language invocations

This call is supported in the following programming languages:

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 (AIX, OS/2, OS/390, Windows NT)

```
call MQCONNX (QMgrName, ConnectOpts, Hconn, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl QMgrName     char(48);       /* Name of queue manager */
dcl ConnectOpts  like MQCNO;     /* Options that control the action of
                                    MQCONNX */
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 (OS/390 only)

```
CALL MQCONNX,(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 MQCONNX
HCONN               DS      F            Connection handle
COMPCODE            DS      F            Completion code
REASON              DS      F            Reason code qualifying CompCode
```

Visual Basic invocation (Windows only)

```
MQCONNX Name, 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 MQCONNX'
Dim Hconn       As Long      'Connection handle'
Dim CompCode    As Long      'Completion code'
Dim Reason      As Long      'Reason code qualifying CompCode'
```

MQCONNX - Language invocations

Chapter 31. 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 OS/390, 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 AS/400, applications running in compatibility mode do not need to issue this call. See "Chapter 29. MQCONN - Connect queue manager" on page 333 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 OS/390 for CICS applications, and on AS/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 OS/390, *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.

© Copyright IBM Corp. 1994, 2000

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_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call reentered 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 489.

Usage notes

1. If an MQDISC call is issued when the application still has objects open, 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 OS/390, 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 indicate that the unit of work should be committed, when asked by the unit-of-work coordinator.

        On OS/390, 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 OS/390, the actions described under (a) above occur.
      - 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 OS/390, 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 MQCONNX 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 and Windows NT, 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.
5. On AS/400, applications running in compatibility mode do not have to issue this call; see the MQCONN call for more details.

Language invocations

This call is supported in the following programming languages:

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 (AIX, OS/2, OS/390, VSE/ESA, Windows NT)

```
call MQDISC (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 (OS/390 only)

```
         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 (Tandem NSK only)

```
INT(32)   .EXT HConn ;
INT(32)   .EXT CC;
INT(32)   .EXT Reason;

CALL MQDISC(HConn, CC, Reason);
```

Visual Basic invocation (Windows only)

```
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'
```

Chapter 32. 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 OS/390 for CICS applications, and on AS/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 34. MQOPEN - Open object" on page 375 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 9. MQMD - Message descriptor" on page 125 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 7. MQGMO - Get-message options" on page 81).

© Copyright IBM Corp. 1994, 2000

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 7. MQGMO - Get-message options" on page 81 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 39. Attributes for queues" on page 427.

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 7. MQGMO - Get-message options" on page 81 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 to perform the conversion of the message data (see "Chapter 7. MQGMO - Get-message options" on page 81 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 7. MQGMO - Get-message options" on page 81 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:

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_INCONSISTENT_CCSIDS
(2243, X'8C3') Message segments have differing CCSIDs.
MQRC_INCONSISTENT_ENCODINGS
(2244, X'8C4') Message segments have differing encodings.
MQRC_NO_MSG_LOCKED
(2209, X'8A1') No message locked.
MQRC_NOT_CONVERTED
(2119, X'847') Message data not 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_CCSID_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_CCSID_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_LOAD_ERROR
(2183, X'887') Unable to load API crossing 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 reentered before previous call complete.
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 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_INCOMPLETE_GROUP
(2241, X'8C1') Message group not complete.
MQRC_INCOMPLETE_MSG
(2242, X'8C2') Logical message not complete.
MQRC_INCONSISTENT_BROWSE
(2259, X'8D3') Inconsistent browse specification.
MQRC_INCONSISTENT_UOW
(2245, X'8C5') Inconsistent unit-of-work specification.
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_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 489.

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 an MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT option is specified on the *GetMsgOpts* parameter (see the *Options* field described in "Chapter 7. MQGMO - Get-message options" on page 81).
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.

   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 9. MQMD - Message descriptor" on page 125 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 or MQGMO_BROWSE_NEXT option is set, in which case the context is marked as not available). This 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 *MQSeries 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 automatically by the queue manager; these are called "built-in" formats:
  MQFMT_ADMIN
  MQFMT_CICS
  MQFMT_COMMAND_1
  MQFMT_COMMAND_2
  MQFMT_DEAD_LETTER_HEADER
  MQFMT_DIST_HEADER
  MQFMT_EVENT
  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_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.
- 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 599 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 599.

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 should be noted:
    - On the MQGET call, 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.

- On the MQGET call, 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.

  On the MQGETANY call, 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 Tandem NonStop Kernel, the following restrictions apply:
    - If the MQGET call is issued outside a Tandem 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 Tandem 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

This call is supported in the following programming languages:

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:

```
**   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 CMQGMOV.
**   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 (AIX, OS/2, OS/390, VSE/ESA, Windows NT)

```
call MQGET (Hconn, Hobj, MsgDesc, GetMsgOpts, BufferLength, Buffer,
           DataLength, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn         fixed bin(31);  /* Connection handle */
dcl Hobj          fixed bin(31);  /* Object handle */
dcl MsgDesc       like MQMD;      /* Message descriptor */
dcl GetMsgOpts    like MQGMO;     /* Options that control the action of
                                     MQGET */
dcl BufferLength  fixed bin(31);  /* Length in bytes of the Buffer
                                     area */
dcl Buffer        char(n);        /* Area to contain the message data */
dcl DataLength    fixed bin(31);  /* Length of the message */
dcl CompCode      fixed bin(31);  /* Completion code */
dcl Reason        fixed bin(31);  /* Reason code qualifying CompCode */
```

System/390 assembler invocation (OS/390 only)

```
          CALL MQGET,(HCONN,HOBJ,MSGDESC,GETMSGOPTS,BUFFERLENGTH,BUFFER,    X
                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 (Tandem NSK only)

```
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;

CALL MQGET(HConn, HObj, MsgDesc, GetMsgOpt, BufferLen, Buffer,
                          DataLen, CC, Reason);
```

Visual Basic invocation (Windows only)

```
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'
```

Chapter 33. MQINQ - Inquire about 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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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 OS/390 for CICS applications, and on AS/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.

© Copyright IBM Corp. 1994, 2000

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 70. 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
MQCA_CLUSTER_NAMELIST	Cluster namelist (MQ_NAMELIST_NAME_LENGTH).	1

Table 70. MQINQ attribute selectors for queues (continued). See the bottom of the table for an explanation of the notes.

Selector	Description	Note
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

Table 70. MQINQ attribute selectors for queues (continued). See the bottom of the table for an explanation of the notes.

Selector	Description	Note
MQIA_TRIGGER_TYPE	Trigger type.	
MQIA_USAGE	Usage.	
Notes: 1. Supported only on AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems. 2. Supported only on AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems. 3. Supported only on OS/390. 4. Not supported on OS/390. 5. Not supported on VSE/ESA.		

Table 71. MQINQ attribute selectors for namelists. See the bottom of Table 70 on page 364 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
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 72. MQINQ attribute selectors for process definitions. See the bottom of Table 70 on page 364 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 73. MQINQ attribute selectors for the queue manager. See the bottom of Table 70 on page 364 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

Table 73. MQINQ attribute selectors for the queue manager (continued). See the bottom of Table 70 on page 364 for an explanation of the notes.

Selector	Description	Note
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
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.	1
MQIA_CHANNEL_AUTO_DEF_EVENT	Control attribute for automatic channel definition events.	1
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_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.	4, 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.

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_LOAD_ERROR
(2183, X'887') Unable to load API crossing exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call reentered before previous call complete.

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.
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 489.

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 76 on page 428 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 *MQSeries 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 39. Attributes for queues" on page 427
   - "Chapter 40. Attributes for namelists" on page 459
   - "Chapter 41. Attributes for process definitions" on page 463
   - "Chapter 42. Attributes for the queue manager" on page 469

Language invocations

This call is supported in the following programming languages:

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.
**   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 (AIX, OS/2, OS/390, VSE/ESA, Windows NT)

```
call MQINQ (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount,
            IntAttrs, CharAttrLength, CharAttrs, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn           fixed bin(31);  /* Connection handle */
dcl Hobj            fixed bin(31);  /* Object handle */
dcl SelectorCount   fixed bin(31);  /* Count of selectors */
dcl Selectors(n)    fixed bin(31);  /* Array of attribute selectors */
dcl IntAttrCount    fixed bin(31);  /* Count of integer attributes */
dcl IntAttrs(n)     fixed bin(31);  /* Array of integer attributes */
dcl CharAttrLength  fixed bin(31);  /* Length of character attributes
                                       buffer */
dcl CharAttrs       char(n);        /* Character attributes */
dcl CompCode        fixed bin(31);  /* Completion code */
dcl Reason          fixed bin(31);  /* Reason code qualifying
                                       CompCode */
```

System/390 assembler invocation (OS/390 only)

```
         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 (Tandem NSK only)

```
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 (Windows only)

```
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'
```

Chapter 34. 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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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 OS/390 for CICS applications, and on AS/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 11. MQOD - Object descriptor" on page 193 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

© Copyright IBM Corp. 1994, 2000

type of the dynamic queue is determined by the value of the *DefinitionType* attribute of the model queue (see “Chapter 39. Attributes for queues” on page 427). 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 74 on page 381).

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 39. Attributes for queues” on page 427 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).

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 7. MQGMO - Get-message options" on page 81 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.

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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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.

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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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 *MQSeries 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 13. MQPMO - Put message options" on page 211.

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 in the following environments: VSE/ESA, Windows 3.1, Windows 95, Windows 98.

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 *MQSeries 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 in the following environments: VSE/ESA, Windows 3.1, Windows 95, Windows 98.

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 *MQSeries 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 in the following environments: VSE/ESA, Windows 3.1, Windows 95, Windows 98.

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 *MQSeries 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 *MQSeries 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

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. This option is accepted but ignored on: Windows 3.1, Windows 95, Windows 98.

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 OS/390, 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.

Table 74. Valid MQOPEN options for each queue type

Option	Alias (note 1)	Local and Model	Remote	Nonlocal Cluster	Distribution list
MQOO_INPUT_AS_Q_DEF	✓	✓	—	—	—
MQOO_INPUT_SHARED	✓	✓	—	—	—
MQOO_INPUT_EXCLUSIVE	✓	✓	—	—	—
MQOO_BROWSE	✓	✓	—	—	—
MQOO_OUTPUT	✓	✓	✓	✓	✓
MQOO_INQUIRE	✓	✓	Note 2	✓	—
MQOO_SET	✓	✓	Note 2	—	—
MQOO_BIND_ON_OPEN (note 3)	✓	✓	✓	✓	✓
MQOO_BIND_NOT_FIXED (note 3)	✓	✓	✓	✓	✓
MQOO_BIND_AS_Q_DEF (note 3)	✓	✓	✓	✓	✓
MQOO_SAVE_ALL_CONTEXT	✓	✓	—	—	—
MQOO_PASS_IDENTITY_CONTEXT	✓	✓	✓	✓	✓
MQOO_PASS_ALL_CONTEXT	✓	✓	✓	✓	✓

Table 74. Valid MQOPEN options for each queue type (continued)

Option	Alias (note 1)	Local and Model	Remote	Nonlocal Cluster	Distribution list
MQOO_SET_IDENTITY_CONTEXT	✓	✓	✓	✓	✓
MQOO_SET_ALL_CONTEXT	✓	✓	✓	✓	✓
MQOO_ALTERNATE_USER_AUTHORITY	✓	✓	✓	✓	✓
MQOO_FAIL_IF_QUIESCING	✓	✓	✓	✓	✓

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 message queuing 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 handle is restricted to the smallest unit of parallel processing supported by the platform on which the application is running; the handle is not valid outside the unit of parallel processing from which the MQOPEN call was issued:

- On Compaq (DIGITAL) OpenVMS, the scope of the handle is the thread issuing the call.
- On PC DOS, the scope of the handle is the system.
- On OS/390, the scope of the handle is:
  - For CICS, the CICS task issuing the call
  - For IMS, the task issuing the call, up to the next syncpoint; this excludes any subtasks of the task
  - For OS/390 batch and TSO, the task issuing the call; this excludes any subtasks of the task
- On OS/2, the scope of the handle is the thread issuing the call.
- On AS/400, the scope of the handle is the job issuing the call.
- On Tandem NonStop Kernel, the scope of the handle is the process.
- On AIX, HP-UX, Sun Solaris, and other UNIX systems, the scope of the handle is the thread issuing the call.
- On VSE/ESA, the scope of the handle is the CICS task issuing the call.
- On Windows 3.1, and for Windows 3.1 applications running on Windows 95, Windows 98, Windows NT, or Win-OS2, the scope of the handle is the process issuing the call.
- On Windows 95, Windows 98 and Windows NT, the scope of the handle is the thread issuing the 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.

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_LOAD_ERROR
(2183, X'887') Unable to load API crossing exit.
MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.
MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call reentered 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_IN_USE
(2346, X'92A') Coupling-facility structure in use.
MQRC_CF_STRUC_LIST_HDR_IN_USE
(2347, X'92B') Coupling-facility 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.

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_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 489.

Usage notes

1. 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 34. MQOPEN - Open object" on page 375.

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:
  - 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
   - (OS/390 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 *MQSeries 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.
       - (OS/390 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 OS/390, 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 OS/390, the queue manager performs security checks only if security is enabled. For more information on security checking, see the *MQSeries for OS/390 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 11. MQOD - Object descriptor" on page 193):
   - 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.

     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, AS/400, Sun Solaris, Windows NT, plus MQSeries 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.

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.

   Cluster queues are supported in the following environments: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

   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 *MQSeries 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 AS/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.

Language invocations

This call is supported in the following programming languages:

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 (AIX, OS/2, OS/390, VSE/ESA, Windows NT)

```
call MQOPEN (Hconn, ObjDesc, Options, Hobj, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn     fixed bin(31);  /* Connection handle */
dcl ObjDesc   like MQOD;      /* Object descriptor */
dcl Options   fixed bin(31);  /* Options that control the action of
                                 MQOPEN */
dcl Hobj      fixed bin(31);  /* Object handle */
dcl CompCode  fixed bin(31);  /* Completion code */
dcl Reason    fixed bin(31);  /* Reason code qualifying CompCode */
```

System/390 assembler invocation (OS/390 only)

```
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
HOBJ                DS      F           Object handle
COMPCODE            DS      F           Completion code
REASON              DS      F           Reason code qualifying CompCode
```

TAL invocation (Tandem NSK only)

```
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 (Windows only)

```
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 35. 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 OS/390 for CICS applications, and on AS/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 9. MQMD - Message descriptor" on page 125 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 10. MQMDE - Message descriptor extension" on page 185 for more details.

© Copyright IBM Corp. 1994, 2000

PutMsgOpts (MQPMO) – input/output

Options that control the action of MQPUT.

See "Chapter 13. MQPMO - Put message options" on page 211 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_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_LOAD_ERROR
(2183, X'887') Unable to load API crossing 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 reentered before previous call complete.

| **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_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_DH_ERROR
(2135, X'857') Distribution 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_HOBJ_ERROR
(2019, X'7E3') Object handle 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.
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_RESPONSE_RECORDS_ERROR
(2156, X'86C') Response records not valid.
MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.
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_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.

For more information on these reason codes, see "Appendix A. Return codes" on page 489.

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 for 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 (DIGITAL) OpenVMS, the application is the thread.
     - On DOS client, the application is the system.
     - On OS/390, the application is:
       - For CICS, the CICS task
       - For IMS, the task
       - For OS/390 batch, the task

- On OS/2, the application is the thread.
- On AS/400, the application is the job.
- On Tandem 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 client and Windows 3.1, the application is the process.
- On Windows NT and Windows 95, Windows 98, the application is the thread.

- The messages all have the same priority.

Additional conditions for 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, AS/400, Sun Solaris, Windows NT, plus MQSeries 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.
  - 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 39. Attributes for queues" on page 427).

   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. In addition, the checks are performed only if a version-2 or later MQMD is used on the MQPUT or MQPUT1 call; the checks are not performed if a version-1 MQMD is used, even if an MQMDE is present at the start of the application message data.

   The following MQ header structures are validated completely by the queue manager: MQDH, MQMDE.

   For other MQ header structures, the queue manager performs some validation, but does not check every field. Structures that are not supported by the local queue manager, and structures following the first MQDLH in the message, are not validated.

   In addition to general checks on the fields in MQ structures, the following conditions must be satisfied:
   - An MQ structure must not be split over two or more segments – the structure must be entirely contained within one segment.
   - 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 one of the following format names:
     MQFMT_ADMIN
     MQFMT_EVENT
     MQFMT_PCF
   - MQ structures 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.)

5. For the Visual Basic programming language, the following points should be noted:
   - On the MQPUT call, 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.
   - On the MQPUT call, 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.
6. On Tandem NonStop Kernel, if the MQPUT call is issued outside a Tandem TMF transaction *without* the MQPMO_NO_SYNCPOINT option, the reason code MQRC_UNIT_OF_WORK_NOT_STARTED is returned.

Language invocations

This call is supported in the following programming languages:

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 (AIX, OS/2, OS/390, VSE/ESA, Windows NT)

```
call MQPUT (Hconn, Hobj, MsgDesc, PutMsgOpts, BufferLength, Buffer,
           CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn         fixed bin(31);  /* Connection handle */
dcl Hobj          fixed bin(31);  /* Object handle */
dcl MsgDesc       like MQMD;      /* Message descriptor */
dcl PutMsgOpts    like MQPMO;     /* Options that control the action of
                                     MQPUT */
dcl BufferLength  fixed bin(31);  /* Length of the message in Buffer */
dcl Buffer        char(n);        /* Message data */
dcl CompCode      fixed bin(31);  /* Completion code */
dcl Reason        fixed bin(31);  /* Reason code qualifying CompCode */
```

System/390 assembler invocation (OS/390 only)

```
          CALL MQPUT,(HCONN,HOBJ,MSGDESC,PUTMSGOPTS,BUFFERLENGTH,BUFFER,    X
                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 (Tandem NSK only)

```
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 (Windows only)

```
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'
```

Chapter 36. 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 OS/390 for CICS applications, and on AS/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 11. MQOD - Object descriptor" on page 193 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 9. MQMD - Message descriptor" on page 125 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 10. MQMDE - Message descriptor extension" on page 185 for more details.

© Copyright IBM Corp. 1994, 2000

PutMsgOpts (MQPMO) – input/output

Options that control the action of MQPUT1.

See "Chapter 13. MQPMO - Put message options" on page 211 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_LOAD_ERROR
(2183, X'887') Unable to load API crossing 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 reentered 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_IN_USE
(2346, X'92A') Coupling-facility structure in use.
MQRC_CF_STRUC_LIST_HDR_IN_USE
(2347, X'92B') Coupling-facility 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_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_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_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_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.
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_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_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_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.

For more information on these reason codes, see "Appendix A. Return codes" on page 489.

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 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 388 for the MQOPEN call, and usage note 3 on page 400 for the MQPUT call.

   Distribution lists are supported in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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 402 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 should be noted:
   - On the MQPUT1 call, 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.
   - On the MQPUT1 call, 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.
8. On Tandem NonStop Kernel, if the MQPUT1 call is issued outside a Tandem TMF transaction *without* the MQPMO_NO_SYNCPOINT option, the reason code MQRC_UNIT_OF_WORK_NOT_STARTED is returned.

Language invocations

This call is supported in the following programming languages:

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 (AIX, OS/2, OS/390, VSE/ESA, Windows NT)

```
call MQPUT1 (Hconn, ObjDesc, MsgDesc, PutMsgOpts, BufferLength, Buffer,
             CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn         fixed bin(31);  /* Connection handle */
dcl ObjDesc       like MQOD;      /* Object descriptor */
dcl MsgDesc       like MQMD;      /* Message descriptor */
dcl PutMsgOpts    like MQPMO;     /* Options that control the action of
                                     MQPUT1 */
dcl BufferLength  fixed bin(31);  /* Length of the message in Buffer */
dcl Buffer        char(n);        /* Message data */
dcl CompCode      fixed bin(31);  /* Completion code */
dcl Reason        fixed bin(31);  /* Reason code qualifying CompCode */
```

System/390 assembler invocation (OS/390 only)

```
         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 (Tandem NSK only)

```
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 (Windows only)

```
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 37. 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 OS/390 for CICS applications, and on AS/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.

© Copyright IBM Corp. 1994, 2000

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 75. 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, AS/400, Sun Solaris, Windows NT, plus MQSeries 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_LOAD_ERROR
: (2183, X'887') Unable to load API crossing exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.
MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call reentered before previous call complete.
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 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 489.

Usage notes

1. 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.
2. The values of attributes can be determined using the MQINQ call; see "Chapter 33. MQINQ - Inquire about object attributes" on page 363 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.
3. 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).
4. 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.
5. If the object being set is a cluster queue, there must be a local instance of the cluster queue for the open to succeed.
6. Changes to attributes resulting from use of the MQSET call do not affect the values of the *AlterationDate* and *AlterationTime* attributes.
7. For more information about object attributes, see:
   - "Chapter 39. Attributes for queues" on page 427
   - "Chapter 40. Attributes for namelists" on page 459
   - "Chapter 41. Attributes for process definitions" on page 463
   - "Chapter 42. Attributes for the queue manager" on page 469

Language invocations

This call is supported in the following programming languages:

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 (AIX, OS/2, OS/390, VSE/ESA, Windows NT)

```
call MQSET (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount,
           IntAttrs, CharAttrLength, CharAttrs, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn           fixed bin(31);  /* Connection handle */
dcl Hobj            fixed bin(31);  /* Object handle */
dcl SelectorCount   fixed bin(31);  /* Count of selectors */
dcl Selectors(n)    fixed bin(31);  /* Array of attribute selectors */
dcl IntAttrCount    fixed bin(31);  /* Count of integer attributes */
dcl IntAttrs(n)     fixed bin(31);  /* Array of integer attributes */
dcl CharAttrLength  fixed bin(31);  /* Length of character attributes
                                       buffer */
dcl CharAttrs       char(n);        /* Character attributes */
```

```
dcl CompCode        fixed bin(31);  /* Completion code */
dcl Reason          fixed bin(31);  /* Reason code qualifying
                                       CompCode */
```

System/390 assembler invocation (OS/390 only)

```
         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 (Tandem NSK only)

```
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 (Windows only)

```
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'
```

Chapter 38. MQSYNC - Synchronize statistics updates (Tandem NSK only)

The MQSYNC call is included in this release of MQSeries for Tandem NonStop Kernel for backwards compatibility with MQSeries for Tandem NSK, Version 1.5.1. but performs no function.

The call always returns a *CompCode* of MQCC_OK, and a *Reason* of MQRC_NONE.

Syntax

MQSYNC *(TransId, CommitAbort, CompCode, Reason)*

Parameters

TransId (MQCHAR48) – input
Transaction identifier.

CommitAbort (MQCHAR48) – input
Commit flag.

CompCode (MQLONG) – output
Completion code.

It is the following:
MQCC_OK
Successful completion.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

For *CompCode* of MQCC_OK:
MQRC_NONE
(0, X'000') No reason to report.

© Copyright IBM Corp. 1994, 2000

Language invocations

This call is supported in the following languages.

C language invocation

```
transaction_id_def TransID;
int         CommitAbort;
MQLONG      CompCode;
MQLONG      Reason;

MQSYNC(&TransID, CommitAbort, &CompCode, &Reason);
```

COBOL language invocation

```
01 TRANSID       NATIVE-4.
01 COMMITABORT   NATIVE-4.
01 COMPCODE      NATIVE-4.
01 REASON        NATIVE-4.

CALL 'MQSYNC' USING TRANSID COMMITABORT.
```

TAL language invocation

```
STRING   .EXT TransID;
INT CommitAbort;
INT(32)  .EXT CC;
INT(32)  .EXT Reason;

CALL MQSYNC(TransID, CommitAbort, CC, Reason);
```

Part 3. Attributes of objects

Chapter 39. Attributes for queues 427
Overview. 428
AlterationDate (MQCHAR12) 430
AlterationTime (MQCHAR8) 430
BackoutRequeueQName (MQCHAR48). . . . 431
BackoutThreshold (MQLONG) 431
BaseQName (MQCHAR48) 431
| CFStrucName (MQCHAR12) 432
ClusterName (MQCHAR48) 432
ClusterNamelist (MQCHAR48) 433
CreationDate (MQCHAR12) 433
CreationTime (MQCHAR8) 433
CurrentQDepth (MQLONG) 434
DefBind (MQLONG) 434
DefinitionType (MQLONG). 435
DefInputOpenOption (MQLONG) 436
DefPersistence (MQLONG) 436
DefPriority (MQLONG) 437
DistLists (MQLONG) 438
HardenGetBackout (MQLONG) 439
IndexType (MQLONG) 440
InhibitGet (MQLONG) 441
InhibitPut (MQLONG) 441
InitiationQName (MQCHAR48) 442
MaxMsgLength (MQLONG) 442
MaxQDepth (MQLONG) 443
MsgDeliverySequence (MQLONG) 443
OpenInputCount (MQLONG) 444
OpenOutputCount (MQLONG) 445
ProcessName (MQCHAR48) 445
QDepthHighEvent (MQLONG) 446
QDepthHighLimit (MQLONG) 446
QDepthLowEvent (MQLONG) 446
QDepthLowLimit (MQLONG). 447
QDepthMaxEvent (MQLONG) 447
QDesc (MQCHAR64). 448
QName (MQCHAR48) 448
QServiceInterval (MQLONG) 449
QServiceIntervalEvent (MQLONG) 449
| QSGDisp (MQLONG) 450
QType (MQLONG) 450
RemoteQMgrName (MQCHAR48) 451
RemoteQName (MQCHAR48). 451
RetentionInterval (MQLONG) 452
Scope (MQLONG). 452
Shareability (MQLONG). 453
StorageClass (MQCHAR8) 454
TriggerControl (MQLONG). 454
TriggerData (MQCHAR64) 454
TriggerDepth (MQLONG) 455
TriggerMsgPriority (MQLONG) 455
TriggerType (MQLONG). 456
Usage (MQLONG) 456
XmitQName (MQCHAR48). 457

Chapter 40. Attributes for namelists 459
Overview. 459
AlterationDate (MQCHAR12) 459
AlterationTime (MQCHAR8) 459
NameCount (MQLONG) 460
NamelistDesc (MQCHAR64) 460
NamelistName (MQCHAR48) 460
Names (MQCHAR48×NameCount) 460
| QSGDisp (MQLONG) 461

Chapter 41. Attributes for process definitions 463
Overview. 463
AlterationDate (MQCHAR12) 463
AlterationTime (MQCHAR8) 463
ApplId (MQCHAR256) 464
ApplType (MQLONG) 464
EnvData (MQCHAR128). 465
ProcessDesc (MQCHAR64) 465
ProcessName (MQCHAR48) 466
| QSGDisp (MQLONG) 466
UserData (MQCHAR128) 467

Chapter 42. Attributes for the queue manager 469
Overview. 469
AlterationDate (MQCHAR12) 470
AlterationTime (MQCHAR8) 470
AuthorityEvent (MQLONG) 471
ChannelAutoDef (MQLONG) 471
ChannelAutoDefEvent (MQLONG) 471
ChannelAutoDefExit (MQCHARn) 472
ClusterWorkloadData (MQCHAR32). 472
ClusterWorkloadExit (MQCHARn) 472
ClusterWorkloadLength (MQLONG) 473
CodedCharSetId (MQLONG) 473
CommandInputQName (MQCHAR48) 474
CommandLevel (MQLONG) 474
DeadLetterQName (MQCHAR48) 476
DefXmitQName (MQCHAR48) 477
DistLists (MQLONG). 477
| IGQPutAuthority (MQLONG). 477
| IGQUserId (MQLONG) 478
InhibitEvent (MQLONG) 479
| IntraGroupQueuing (MQLONG) 479
LocalEvent (MQLONG) 480
MaxHandles (MQLONG) 480
MaxMsgLength (MQLONG) 480
MaxPriority (MQLONG) 481
MaxUncommittedMsgs (MQLONG) 481
PerformanceEvent (MQLONG) 482
Platform (MQLONG). 482
QMgrDesc (MQCHAR64) 483
QMgrIdentifier (MQCHAR48) 483
QMgrName (MQCHAR48) 483
| QSGName (MQCHAR4). 484
RemoteEvent (MQLONG) 484
RepositoryName (MQCHAR48) 484
RepositoryNamelist (MQCHAR48) 485

© Copyright IBM Corp. 1994, 2000

Object attributes

StartStopEvent (MQLONG). 485
SyncPoint (MQLONG) 485
TriggerInterval (MQLONG). 486

Chapter 39. 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 OS/390.

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.

Alias queue
This is not a physical queue – it is an alternative name for a local queue. The name of the local 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 and remove messages from alias queues – the messages are actually placed on and removed from the local queue to which the alias resolves. The value of the *QType* queue attribute is MQQT_ALIAS.

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.

This type of 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 *MQSeries Intercommunication* book.
- Queue-manager aliasing

© Copyright IBM Corp. 1994, 2000

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 *MQSeries Intercommunication* book.

Applications connected to the local queue manager can place messages on remote queues – 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.

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 indicated by the ✓ symbol in Table 76 and subsequent tables.

Overview

Table 76 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 *MQSeries MQSC Command Reference* for details.

Table 76. Attributes for queues. The columns apply as follows:

- The column for local queues applies also to shared queues.
- 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
AlterationDate	Date when definition was last changed	✓		✓	✓		430
AlterationTime	Time when definition was last changed	✓		✓	✓		430
BackoutRequeueQName	Excessive backout requeue queue name	✓	✓				431
BackoutThreshold	Backout threshold	✓	✓				431
BaseQName	Queue name to which alias resolves			✓			431
CFStrucName	Coupling-facility structure name	✓	✓				432
ClusterName	Name of cluster to which queue belongs	✓		✓	✓		432
ClusterNamelist	Name of namelist object containing names of clusters to which queue belongs	✓		✓	✓		433
CreationDate	Date the queue was created	✓					433

Table 76. Attributes for queues (continued). The columns apply as follows:

- The column for local queues applies also to shared queues.
- 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
CreationTime	Time the queue was created	✓					433
CurrentQDepth	Current queue depth	✓					434
DefBind	Default binding	✓		✓	✓	✓	434
DefinitionType	Queue definition type	✓	✓				435
DefInputOpenOption	Default input open option	✓	✓				436
DefPersistence	Default message persistence	✓	✓	✓	✓	✓	436
DefPriority	Default message priority	✓	✓	✓	✓	✓	437
DistLists	Distribution list support	✓	✓				438
HardenGetBackout	Whether to maintain an accurate backout count	✓	✓				439
IndexType	Index type	✓	✓				440
InhibitGet	Controls whether get operations for the queue are allowed	✓	✓	✓			441
InhibitPut	Controls whether put operations for the queue are allowed	✓	✓	✓	✓	✓	441
InitiationQName	Name of initiation queue	✓	✓				442
MaxMsgLength	Maximum message length in bytes	✓	✓				442
MaxQDepth	Maximum queue depth	✓	✓				443
MsgDeliverySequence	Message delivery sequence	✓	✓				443
OpenInputCount	Number of opens for input	✓					444
OpenOutputCount	Number of opens for output	✓					445
ProcessName	Process name	✓	✓				445
QDepthHighEvent	Controls whether Queue Depth High events are generated	✓	✓				446
QDepthHighLimit	High limit for queue depth	✓	✓				446
QDepthLowEvent	Controls whether Queue Depth Low events are generated	✓	✓				446
QDepthLowLimit	Low limit for queue depth	✓	✓				447
QDepthMaxEvent	Controls whether Queue Full events are generated	✓	✓				447
QDesc	Queue description	✓	✓	✓	✓	✓	448
QName	Queue name	✓		✓	✓	✓	448
QServiceInterval	Target for queue service interval	✓	✓				449
QServiceIntervalEvent	Controls whether Service Interval High or Service Interval OK events are generated	✓	✓				449
QSGDisp	Queue-sharing group disposition	✓		✓	✓		450
QType	Queue type	✓		✓	✓	✓	450
RemoteQMgrName	Name of remote queue manager				✓		451
RemoteQName	Name of remote queue				✓		451
RetentionInterval	Retention interval	✓	✓				452

Table 76. Attributes for queues (continued). The columns apply as follows:

- The column for local queues applies also to shared queues.
- 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
Scope	Controls whether an entry for the queue also exists in a cell directory	✓		✓	✓		452
Shareability	Queue shareability	✓	✓				453
StorageClass	Storage class for queue	✓	✓				454
TriggerControl	Trigger control	✓	✓				454
TriggerData	Trigger data	✓	✓				454
TriggerDepth	Trigger depth	✓	✓				455
TriggerMsgPriority	Threshold message priority for triggers	✓	✓				455
TriggerType	Trigger type	✓	✓				456
Usage	Queue usage	✓	✓				456
XmitQName	Transmission queue name				✓		457

AlterationDate (MQCHAR12)

Date when definition was last changed.

Local	Model	Alias	Remote	Cluster
✓		✓	✓	

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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

AlterationTime (MQCHAR8)

Time when definition was last changed.

Local	Model	Alias	Remote	Cluster
✓		✓	✓	

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 OS/390, 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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

BackoutRequeueQName (MQCHAR48)

Excessive backout requeue queue name.

Local	Model	Alias	Remote	Cluster
✓	✓			

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
✓	✓			

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.

BaseQName (MQCHAR48)

The queue name to which the alias resolves.

Local	Model	Alias	Remote	Cluster
		✓		

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
✓	✓			

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 OS/390.

ClusterName (MQCHAR48)

Name of cluster to which queue belongs.

Local	Model	Alias	Remote	Cluster
✓		✓	✓	

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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

ClusterNamelist (MQCHAR48)

Name of namelist object containing names of clusters to which queue belongs.

Local	Model	Alias	Remote	Cluster
✓		✓	✓	

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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

CreationDate (MQCHAR12)

Date when queue was created.

Local	Model	Alias	Remote	Cluster
✓				

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, `1992-09-23bb`, where `bb` represents 2 blank characters).

- On AS/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.

CreationTime (MQCHAR8)

Time when queue was created.

Local	Model	Alias	Remote	Cluster
✓				

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`). The time is local time.

- On OS/390, the time is Greenwich Mean Time (GMT), subject to the system clock being set accurately to GMT.

- On AS/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
✓				

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 9. MQMD - Message descriptor" on page 125.

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
✓		✓	✓	✓

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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

DefinitionType (MQLONG)

Queue definition type.

Local	Model	Alias	Remote	Cluster
✓	✓			

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 42. Attributes for the queue manager" on page 469).

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 27. MQCLOSE - Close object" on page 319 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.

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 27. MQCLOSE - Close object" on page 319 for more details.

The value of the *QSGDisp* attribute for a shared dynamic queue is MQQSGD_SHARED.

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
✓	✓			

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 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
✓	✓	✓	✓	✓

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

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.

DefPriority (MQLONG)

Default message priority

Local	Model	Alias	Remote	Cluster
✓	✓	✓	✓	✓

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 39. Attributes for queues" on page 427 for more information.

Priorities are in the range zero (lowest) through *MaxPriority* (highest); see the *MaxPriority* attribute described in "Chapter 42. Attributes for the queue manager" on page 469.

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
✓	✓			

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, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

HardenGetBackout (MQLONG)

Whether to maintain an accurate backout count.

Local	Model	Alias	Remote	Cluster
✓	✓			

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.

- On Compaq (DIGITAL) OpenVMS, OS/2, AS/400, Tandem NonStop Kernel, UNIX systems, and Windows NT, 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
✓	✓			

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.

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.

The index type that should be used in various cases in shown in Table 77.

Table 77. Recommended values of queue index type

Selection criteria on MQGET call	Index type for nonshared queue	Index type for shared queue
None	Any	Any
Message identifier	MQIT_MSG_ID recommended	MQIT_MSG_ID required
Correlation identifier	MQIT_CORREL_ID recommended	MQIT_CORREL_ID required
Message identifier plus correlation identifier	MQIT_MSG_ID or MQIT_CORREL_ID recommended	MQIT_MSG_ID or MQIT_CORREL_ID required

Table 77. Recommended values of queue index type (continued)

Selection criteria on MQGET call	Index type for nonshared queue	Index type for shared queue
Message token	MQIT_MSG_TOKEN 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 OS/390.

InhibitGet (MQLONG)

Controls whether get operations for this queue are allowed.

Local	Model	Alias	Remote	Cluster
✓	✓	✓		

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
✓	✓	✓	✓	✓

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
✓	✓			

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 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
✓	✓			

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 reason code:

- MQRC_MSG_TOO_BIG_FOR_Q if the message to too big for the queue

- MQRC_MSG_TOO_BIG_FOR_Q_MGR if the message to 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 OS/390:
  - 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, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems, the maximum message length is 100 MB (104 857 600 bytes).
- On Compaq (DIGITAL) OpenVMS, Tandem NonStop Kernel, UNIX systems not listed above, Windows 3.1, Windows 95, Windows 98, plus MQSeries 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 35. MQPUT - Put message" on page 393.

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
✓	✓			

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 Compaq (DIGITAL) OpenVMS, OS/2, AS/400, Tandem NonStop Kernel, UNIX systems, and Windows NT, 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
✓	✓			

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).

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 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
✓				

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.

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
✓				

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
✓	✓			

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.

This attribute is not supported in the following environments: Windows 3.1, Windows 95, Windows 98.

QDepthHighEvent (MQLONG)

Controls whether Queue Depth High events are generated.

Local	Model	Alias	Remote	Cluster
✓	✓			

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.

The value is one of the following:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see the *MQSeries Event Monitoring* book.

To determine the value of this attribute, use the MQIA_Q_DEPTH_HIGH_EVENT selector with the MQINQ call.

- On OS/390, the MQINQ call cannot be used to determine the value of this attribute.

QDepthHighLimit (MQLONG)

High limit for queue depth.

Local	Model	Alias	Remote	Cluster
✓	✓			

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 not supported in the following environments: Windows 3.1, Windows 95, Windows 98.
- This attribute is supported on OS/390, but the MQINQ call cannot be used to determine its value.

QDepthLowEvent (MQLONG)

Controls whether Queue Depth Low events are generated.

Local	Model	Alias	Remote	Cluster
✓	✓			

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 *MQSeries Event Monitoring* book.

To determine the value of this attribute, use the MQIA_Q_DEPTH_LOW_EVENT selector with the MQINQ call.

- On OS/390, the MQINQ call cannot be used to determine the value of this attribute.

QDepthLowLimit (MQLONG)

Low limit for queue depth.

Local	Model	Alias	Remote	Cluster
✓	✓			

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 not supported in the following environments: Windows 3.1, Windows 95, Windows 98.
- This attribute is supported on OS/390, but the MQINQ call cannot be used to determine its value.

QDepthMaxEvent (MQLONG)

Controls whether Queue Full events are generated.

Local	Model	Alias	Remote	Cluster
✓	✓			

Overview

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 *MQSeries Event Monitoring* book.

To determine the value of this attribute, use the MQIA_Q_DEPTH_MAX_EVENT selector with the MQINQ call.

- On OS/390, the MQINQ call cannot be used to determine the value of this attribute.

QDesc (MQCHAR64)

Queue description.

Local	Model	Alias	Remote	Cluster
✓	✓	✓	✓	✓

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
✓		✓	✓	✓

This is the name of a queue defined on the local queue manager. For more information about queue names, see the *MQSeries 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
✓	✓			

This is the service interval used for comparison to generate Service Interval High and Service Interval OK events. See the *QServiceIntervalEvent* 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 not supported in the following environments: Windows 3.1, Windows 95, Windows 98.
- This attribute is supported on OS/390, but the MQINQ call cannot be used to determine its value.

QServiceIntervalEvent (MQLONG)

Controls whether Service Interval High or Service Interval OK events are generated.

Local	Model	Alias	Remote	Cluster
✓	✓			

- 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 *MQSeries Event Monitoring* book.

To determine the value of this attribute, use the MQIA_Q_SERVICE_INTERVAL_EVENT selector with the MQINQ call.

On OS/390, the MQINQ call cannot be used to determine the value of this attribute.

QSGDisp (MQLONG)

Queue-sharing group disposition.

Local	Model	Alias	Remote	Cluster
✓		✓	✓	

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 OS/390.

QType (MQLONG)

Queue type.

Local	Model	Alias	Remote	Cluster
✓		✓	✓	✓

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
			✓	

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
			✓	

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*.

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
✓	✓			

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
✓		✓	✓	

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.

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 AS/400, the attribute is supported, but only MQSCO_Q_MGR is valid.
- On OS/390, Windows 3.1, and Windows 95, Windows 98, the attribute is not supported.

Shareability (MQLONG)

Whether queue can be shared for input.

Local	Model	Alias	Remote	Cluster
✓	✓			

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
✓	✓			

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.

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 OS/390.

TriggerControl (MQLONG)

Trigger control.

Local	Model	Alias	Remote	Cluster
✓	✓			

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.

This attribute is not supported in the following environments: Windows 3.1, Windows 95, Windows 98.

TriggerData (MQCHAR64)

Trigger data.

Local	Model	Alias	Remote	Cluster
✓	✓			

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
✓	✓			

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.

This attribute is not supported in the following environments: Windows 3.1, Windows 95, Windows 98.

TriggerMsgPriority (MQLONG)

Threshold message priority for triggers.

Local	Model	Alias	Remote	Cluster
✓	✓			

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 42. Attributes for the queue manager" on page 469); 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.

This attribute is not supported in the following environments: Windows 3.1, Windows 95, Windows 98.

TriggerType (MQLONG)

Trigger type.

Local	Model	Alias	Remote	Cluster
✓	✓			

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.

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.

This attribute is not supported in the following environments: Windows 3.1, Windows 95, Windows 98.

Usage (MQLONG)

Queue usage.

Local	Model	Alias	Remote	Cluster
✓	✓			

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 *MQSeries 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 23. MQXQH - Transmission queue header" on page 291), 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
			✓	

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 40. Attributes for namelists

Namelists are supported in the following environments: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Overview

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 *MQSeries MQSC Command Reference* for details.

Table 78. Attributes for namelists

Attribute	Description	Page
AlterationDate	Date when definition was last changed	459
AlterationTime	Time when definition was last changed	459
NameCount	Number of names in namelist	460
NamelistDesc	Namelist description	460
NamelistName	Namelist name	460
Names	A list of *NameCount* names	460
QSGDisp	Queue-sharing group disposition	461

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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

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.

© Copyright IBM Corp. 1994, 2000

This attribute is supported in the following environments: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

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 *MQSeries 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).

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.

Names (MQCHAR48×NameCount)

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 *MQSeries 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.

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 OS/390.

Chapter 41. Attributes for process definitions

Process definitions are not supported in the following environments: VSE/ESA, Windows 3.1, Windows 95, Windows 98.

Overview

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 *MQSeries MQSC Command Reference* for details.

Table 79. Attributes for process definitions

Attribute	Description	Page
AlterationDate	Date when definition was last changed	463
AlterationTime	Time when definition was last changed	463
ApplId	Application identifier	464
ApplType	Application type	464
EnvData	Environment data	465
ProcessDesc	Process description	465
ProcessName	Process name	466
QSGDisp	Queue-sharing group disposition	466
UserData	User data	467

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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

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`.

© Copyright IBM Corp. 1994, 2000

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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

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 OS/390, *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 DOS client, OS/2, and Windows systems, the program name can be prefixed with a drive and directory path.
- 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.

ApplType (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.

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
DOS client application.

MQAT_IMS
IMS application.

MQAT_MVS
OS/390 or TSO application (same value as MQAT_OS390).
MQAT_NOTES_AGENT
Lotus Notes Agent application.
MQAT_NSK
Tandem NonStop Kernel application.
MQAT_OS2
OS/2 or Presentation Manager application.
MQAT_OS400
AS/400 application.
MQAT_UNIX
UNIX application.
MQAT_VMS
Digital OpenVMS application.
MQAT_WINDOWS
Windows client, Windows 3.1 application.
MQAT_WINDOWS_NT
Windows NT, Windows 95, Windows 98 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 structure, followed by one blank, followed by *EnvData* with trailing blanks removed. The following notes apply to the environments indicated:

- On OS/390, *EnvData* is not used by the trigger-monitor applications provided by MQSeries.
- 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.

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 OS/390.

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 42. 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 `ALTER QMGR` MQSC command. All can be inquired by opening a special MQOT_Q_MGR object, and using the MQINQ call with the handle returned. The attributes can also all be displayed by using the `DISPLAY QMGR` command.

Overview

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 *MQSeries MQSC Command Reference* for details.

Table 80. Attributes for the queue manager

Attribute	Description	Page
AlterationDate	Date when definition was last changed	470
AlterationTime	Time when definition was last changed	470
AuthorityEvent	Controls whether authorization (Not Authorized) events are generated	471
ChannelAutoDef	Controls whether automatic channel definition is permitted	471
ChannelAutoDefEvent	Controls whether channel automatic-definition events are generated	471
ChannelAutoDefExit	Name of user exit for automatic channel definition	472
ClusterWorkloadData	User data for cluster workload exit	472
ClusterWorkloadExit	Name of user exit for cluster workload management	472
ClusterWorkloadLength	Maximum length of message data passed to cluster workload exit	473
CodedCharSetId	Coded character set identifier	473
CommandInputQName	Command input queue name	474
CommandLevel	Command level	474
DeadLetterQName	Name of dead-letter queue	476
DefXmitQName	Default transmission queue name	477
DistLists	Distribution list support	477
IGQPutAuthority	Intra-group queuing put authority	477
IGQUserId	Intra-group queuing user identifier	478
InhibitEvent	Controls whether inhibit (Inhibit Get and Inhibit Put) events are generated	479
IntraGroupQueuing	Intra-group queuing support	479

© Copyright IBM Corp. 1994, 2000

Table 80. Attributes for the queue manager (continued)

Attribute	Description	Page
LocalEvent	Controls whether local error events are generated	480
MaxHandles	Maximum number of handles	480
MaxMsgLength	Maximum message length in bytes	480
MaxPriority	Maximum priority	481
MaxUncommittedMsgs	Maximum number of uncommitted messages within a unit of work	481
PerformanceEvent	Controls whether performance-related events are generated	482
Platform	Platform on which the queue manager is running	482
QMgrDesc	Queue manager description	483
QMgrIdentifier	Unique internally-generated identifier of queue manager	483
QMgrName	Queue manager name	483
QSGName	Name of queue-sharing group	484
RemoteEvent	Controls whether remote error events are generated	484
RepositoryName	Name of cluster for which this queue manager provides repository services	484
RepositoryNamelist	Name of namelist object containing names of clusters for which this queue manager provides repository services	485
StartStopEvent	Controls whether start and stop events are generated	485
SyncPoint	Syncpoint availability	485
TriggerInterval	Trigger-message interval	486

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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

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 *MQSeries Event Monitoring* book.

To determine the value of this attribute, use the MQIA_AUTHORITY_EVENT selector with the MQINQ call.

- On OS/390, 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. The value is one of the following:

MQCHAD_DISABLED
Channel auto-definition disabled.

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, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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 *MQSeries 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, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems. On OS/390, the MQINQ call cannot be used to determine the value of this attribute.

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 *MQSeries 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.

This attribute is supported in the following environments: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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 *MQSeries 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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

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 OS/390, the value is set from the system parameters when the queue manager is started; the default value is 500. Refer to the *MQSeries for OS/390 System Setup Guide* for further information.
- On OS/2 and Windows NT, the value is the primary CODEPAGE of the user creating the queue manager.
- On AS/400, the value is that which is set in the environment when the queue manager is first created.
- On Compaq (DIGITAL) OpenVMS, Tandem 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 OS/390, the name of the queue is SYSTEM.COMMAND.INPUT, and only MQSC commands can be sent to it. Refer to the *MQSeries 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 *MQSeries Programmable System Management*. 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™
  - 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 OS/390 Version 2 Release 2.0.
- 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

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 Solaris Version 5 Release 1
- MQSeries for Windows NT Version 5 Release 1

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.

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 9. MQMD - Message descriptor" on page 125) 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 6. MQDLH - Dead-letter header" on page 69 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 MQSeries 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.

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, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

IGQPutAuthority (MQLONG)

Intra-group queuing put authority.

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.

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 OS/390.

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 *MQSeries Event Monitoring* book.

To determine the value of this attribute, use the MQIA_INHIBIT_EVENT selector with the MQINQ call.

- On OS/390, the MQINQ call cannot be used to determine the value of this attribute.

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.

Messages destined for other queue managers in the queue-sharing group are transmitted using the shared transmission queue if both of the following conditions are satisfied:

- The message is not persistent.
- 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 these conditions are 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 OS/390.

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 *MQSeries Event Monitoring* book.

To determine the value of this attribute, use the MQIA_LOCAL_EVENT selector with the MQINQ call.

- On OS/390, the MQINQ call cannot be used to determine the value of this attribute.

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 OS/390, "task" means a CICS task, an MVS task, or an IMS dependent region.

The value is in the range 1 through 999 999 999. The default value is determined by the environment:

- On OS/390, 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, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems, the maximum message length is 100 MB (104 857 600 bytes).
- On Compaq (DIGITAL) OpenVMS, Tandem NonStop Kernel, UNIX systems not listed above, Windows 3.1, Windows 95, Windows 98, plus MQSeries 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.

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.

On OS/390, the MQINQ call cannot be used to determine the value of this attribute.

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 *MQSeries Event Monitoring* book.

To determine the value of this attribute, use the MQIA_PERFORMANCE_EVENT selector with the MQINQ call.

- On OS/390, the MQINQ call cannot be used to determine the value of this attribute.

Platform (MQLONG)

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_OS390).

MQPL_NSK
Tandem NonStop Kernel.

MQPL_OS2
OS/2.

MQPL_OS390
OS/390.

MQPL_OS400
AS/400.

MQPL_UNIX
UNIX systems.

MQPL_VMS
Compaq (DIGITAL) OpenVMS.

MQPL_WINDOWS
Windows 3.1.

MQPL_WINDOWS_NT
Windows NT or Windows 95, Windows 98.

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 OS/390, the default value is:

  ```
  MQSeries for OS/390 Vx.y.z
  ```

  where `x`, `y`, and `z` are replaced by the version, release, and modification numbers, respectively.
- 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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries 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 9. MQMD - Message descriptor" on page 125). 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 OS/390, 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 OS/390.

RemoteEvent (MQLONG)

Controls whether remote 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 *MQSeries Event Monitoring* book.

To determine the value of this attribute, use the MQIA_REMOTE_EVENT selector with the MQINQ call.

- On OS/390, the MQINQ call cannot be used to determine the value of this attribute.

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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

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 *MQSeries Event Monitoring* book.

To determine the value of this attribute, use the MQIA_START_STOP_EVENT selector with the MQINQ call.

- On OS/390, the MQINQ call cannot be used to determine the value of this attribute.

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 OS/390, 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 *MQSeries 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.

This attribute is not supported in the following environments: Windows 3.1, Windows 95, Windows 98.

Part 4. Appendixes

© Copyright IBM Corp. 1994, 2000

Appendix A. Return codes

This book contains the return codes associated with the MQI and the MQAI. The return codes associated with:

- Programmable Command Format (PCF) commands are listed in the *MQSeries Programmable System Management* book.
- C++ commands are listed in the *MQSeries Using C++* book.

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.

Reason codes

The reason code parameter (*Reason*) is a qualification to the completion code parameter (*CompCode*).

© Copyright IBM Corp. 1994, 2000

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

The following is a list of reason codes, in alphabetic order, with more detail than is given in the call descriptions. See "MQRC_* (Reason code)" on page 573 for a list of reason codes in numeric order.

MQRC_ADAPTER_CONN_LOAD_ERROR
(2129, X'0851')

Explanation: On an MQCONN call, the connection handling module (CSQBCON for batch and CSQQCONN for IMS) could not be loaded, so the adapter could not link to it.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the correct library concatenation has been specified in the batch application program execution JCL, and in the MQSeries startup JCL.

MQRC_ADAPTER_CONV_LOAD_ERROR
(2133, X'0855')

Explanation: On an MQGET call, the adapter (batch or IMS) could not load the data conversion services modules.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the correct library concatenation has been specified in the batch application program execution JCL, and in the MQSeries startup JCL.

MQRC_ADAPTER_DEFS_ERROR
(2131, X'0853')

Explanation: On an MQCONN call, the subsystem definition module (CSQBDEFV for batch and CSQQDEFV for IMS) does not contain the required control block identifier.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Check your library concatenation. If this is correct, check that the CSQBDEFV or CSQQDEFV module contains the required subsystem ID.

MQRC_ADAPTER_DEFS_LOAD_ERROR
(2132, X'0854')

Explanation: On an MQCONN call, the subsystem definition module (CSQBDEFV for batch and CSQQDEFV for IMS) could not be loaded.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the correct library concatenation has been specified in the application program execution JCL, and in the MQSeries startup JCL.

MQRC_ADAPTER_DISC_LOAD_ERROR
(2138, X'085A')

Explanation: On an MQDISC call, the disconnect handling module (CSQBDSC for batch and CSQQDISC for IMS) could not be loaded, so the adapter could not link to it.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the correct library concatenation has been specified in the application program execution JCL, and in the MQSeries startup JCL. Any uncommitted changes in a unit of work should be backed out. A unit of work that is

coordinated by the queue manager is backed out automatically.

MQRC_ADAPTER_NOT_AVAILABLE (2204, X'089C')

Explanation: This is issued only for CICS applications, if any call is issued and the CICS adapter (a Task Related User Exit) has been disabled, or has not been enabled.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: The application should tidy up and terminate. Any uncommitted changes in a unit of work should be backed out. A unit of work that is coordinated by the queue manager is backed out automatically.

MQRC_ADAPTER_SERV_LOAD_ERROR (2130, X'0852')

Explanation: On an MQI call, the batch adapter could not load the API service module CSQBSRV, and so could not link to it.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the correct library concatenation has been specified in the batch application program execution JCL, and in the MQSeries startup JCL.

MQRC_ADAPTER_STORAGE_SHORTAGE (2127, X'084F')

Explanation: On an MQCONN call, the adapter was unable to acquire storage.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Notify the system programmer. The system programmer should determine why the system is short on storage, and take appropriate action, for example, increase the region size on the step or job card.

MQRC_ALIAS_BASE_Q_TYPE_ERROR (2001, X'07D1')

Explanation: An MQOPEN or MQPUT1 call was issued specifying an alias queue as the destination, but the *BaseQName* in the alias queue definition resolves to a queue that is not a local queue, a local definition of a remote queue, or a cluster queue.

Completion Code: MQCC_FAILED

Programmer Response: Correct the queue definitions.

MQRC_ALREADY_CONNECTED (2002, X'07D2')

Explanation: An MQCONN or MQCONNX call was issued, but the application is already connected to the queue manager.

- On OS/390, this reason code occurs for batch and IMS applications only; it does not occur for CICS applications.
- On Windows NT, MTS objects do not receive this reason code, as additional connections to the queue manager are allowed.

Completion Code: MQCC_WARNING

Programmer Response: None. The *Hconn* parameter returned has the same value as was returned for the previous MQCONN or MQCONNX call.

An MQCONN or MQCONNX call that returns this reason code does *not* mean that an additional MQDISC call must be issued in order to disconnect from the queue manager. If this reason code is returned because the application has been called in a situation where the connect has already been done, a corresponding MQDISC should *not* be issued, because this will cause the application that issued the original MQCONN or MQCONNX call to be disconnected as well.

MQRC_ANOTHER_Q_MGR_CONNECTED (2103, X'0837')

Explanation: An MQCONN or MQCONNX call was issued, but the thread or process is already connected to a different queue manager. The thread or process can connect to only one queue manager at a time.

- On OS/390, this reason code does not occur.
- On Windows NT, MTS objects do not receive this reason code, as connections to other queue managers are allowed.

Completion Code: MQCC_FAILED

Programmer Response: Use the MQDISC call to disconnect from the queue manager that is already connected, and then issue the MQCONN or MQCONNX call to connect to the new queue manager.

Disconnecting from the existing queue manager will close any queues that are currently open; it is recommended that any uncommitted units of work should be committed or backed out before the MQDISC call is issued.

MQRC_API_EXIT_LOAD_ERROR (2183, X'0887')

Explanation: The API crossing exit module could not be linked. If this reason is returned when the API crossing exit is invoked *after* the call has been executed, the call itself may have executed correctly.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the correct library concatenation has been specified, and that the API crossing exit module is executable and correctly named. Any uncommitted changes in a unit of work should be backed out. A unit of work that is coordinated by the queue manager is backed out automatically.

MQRC_APPL_FIRST
(900, X'0384')

Explanation: This is the lowest value for an application-defined reason code returned by a data-conversion exit. Data-conversion exits can return reason codes in the range MQRC_APPL_FIRST through MQRC_APPL_LAST to indicate particular conditions that the exit has detected.

Completion Code: MQCC_WARNING or MQCC_FAILED

Programmer Response: As defined by the writer of the data-conversion exit.

MQRC_APPL_LAST
(999, X'03E7')

Explanation: This is the highest value for an application-defined reason code returned by a data-conversion exit. Data-conversion exits can return reason codes in the range MQRC_APPL_FIRST through MQRC_APPL_LAST to indicate particular conditions that the exit has detected.

Completion Code: MQCC_WARNING or MQCC_FAILED

Programmer Response: As defined by the writer of the data-conversion exit.

MQRC_ASID_MISMATCH
(2157, X'086D')

Explanation: On any MQI call, the caller's primary ASID was found to be different from the home ASID.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Correct the application (MQI calls cannot be issued in cross-memory mode). Any uncommitted changes in a unit of work should be backed out. A unit of work that is coordinated by the queue manager is backed out automatically.

MQRC_BACKED_OUT
(2003, X'07D3')

Explanation: The current unit of work encountered a fatal error or was backed out. This occurs in the following cases:

- On an MQCMIT or MQDISC call, when the commit operation has failed and the unit of work has been backed out. All resources that participated in the unit of work have been returned to their state at the start of the unit of work. The MQCMIT call returns completion code MQCC_FAILED; the MQDISC call returns completion code MQCC_WARNING.
  - On OS/390, this reason code occurs only for batch applications.
- On an MQGET, MQPUT, or MQPUT1 call that is operating within a unit of work, when the unit of work has already encountered an error that prevents the unit of work being committed (for example, when the log space is exhausted). The application must issue the appropriate call to back out the unit of work. For a unit of work coordinated by the queue manager, this call is the MQBACK call, although the MQCMIT call has the same effect in these circumstances.
  - On OS/390, this case does not occur.

Completion Code: MQCC_WARNING or MQCC_FAILED

Programmer Response: Check the returns from previous calls to the queue manager. For example, a previous MQPUT call may have failed.

MQRC_BAG_CONVERSION_ERROR
(2303, X'08FF')

Explanation: The mqBufferToBag or mqGetBag call was issued, but the data in the buffer or message could not be converted into a bag. This occurs when the data to be converted is not valid PCF.

Completion Code: MQCC_FAILED

Programmer Response: Check the logic of the application that created the buffer or message to ensure that the buffer or message contains valid PCF.

If the message contains PCF that is not valid, the message cannot be retrieved using the mqGetBag call:
- If one of the MQGMO_BROWSE_* options was specified, the message remains on the queue and can be retrieved using the MQGET call.
- In other cases, the message has already been removed from the queue and discarded. If the message was retrieved within a unit of work, the unit of work can be backed out and the message retrieved using the MQGET call.

MQRC_BAG_WRONG_TYPE
(2326, X'0916')

Explanation: The *Bag* parameter specifies the handle of a bag that has the wrong type for the call. The bag must be an administration bag, that is, it must be created with the MQCBO_ADMIN_BAG option specified on the mqCreateBag call.

Completion Code: MQCC_FAILED

Programmer Response: Specify the MQCBO_ADMIN_BAG option when the bag is created.

MQRC_BO_ERROR
(2134, X'0856')

Explanation: On an MQBEGIN call, the begin-options structure MQBO is not valid, for one of the following reasons:

- The *StrucId* field is not MQBO_STRUC_ID.
- The *Version* field is not MQBO_VERSION_1.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The queue manager cannot copy the changed structure to application storage, even though the call is successful. This can occur, for example, if the pointer points to read-only storage.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Correct the definition of the MQBO structure. Ensure that required input fields are set correctly.

MQRC_BRIDGE_STARTED
(2125, X'084D')

Explanation: The IMS bridge has been started.

Completion Code: MQCC_WARNING

Programmer Response: None. This reason code is only used to identify the corresponding event message.

MQRC_BRIDGE_STOPPED
(2126, X'084E')

Explanation: The IMS bridge has been stopped.

Completion Code: MQCC_WARNING

Programmer Response: None. This reason code is only used to identify the corresponding event message.

MQRC_BUFFER_ERROR
(2004, X'07D4')

Explanation: The *Buffer* parameter is not valid for one of the following reasons:

- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The parameter pointer points to storage that cannot be accessed for the entire length specified by *BufferLength*.
- For calls where *Buffer* is an output parameter: the parameter pointer points to read-only storage.

Completion Code: MQCC_FAILED

Programmer Response: Correct the parameter.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'07D5')

Explanation: The *BufferLength* parameter is not valid, or the parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

This reason can also be returned to an MQ client program on the MQCONN or MQCONNX call if the negotiated maximum message size for the channel is smaller than the fixed part of any call structure.

Completion Code: MQCC_FAILED

Programmer Response: Specify a value that is zero or greater. For the mqAddString and mqSetString calls, the special value MQBL_NULL_TERMINATED is also valid.

MQRC_CALL_IN_PROGRESS
(2219, X'08AB')

Explanation: The application issued an MQI call whilst another MQI call was already being processed for that connection. Only one call per application connection can be processed at a time.

Concurrent calls can arise only in certain specialized situations, such as in an exit invoked as part of the processing of an MQI call. For example, the data-conversion exit may be invoked as part of the processing of the MQGET call.

- On OS/390, concurrent calls can arise only with batch or IMS applications; an example is when a subtask ends while an MQI call is in progress (for example, an MQGET that is waiting), and there is an end-of-task exit routine that issues another MQI call.
- On OS/2, Windows client, and Windows NT, concurrent calls can also arise if an MQI call is issued in response to a user message while another MQI call is in progress.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that an MQI call cannot be issued while another one is active. Do not issue MQI calls from within a data-conversion exit.

- On OS/390, if you want to provide a subtask to allow an application that is waiting for a message to arrive to be canceled, wait for the message by using MQGET with MQGMO_SET_SIGNAL, rather than MQGMO_WAIT.

MQRC_CD_ERROR
(2277, X'08E5')

Explanation: An MQCONNX call was issued to connect to a queue manager, but the MQCD channel definition structure addressed by the *ClientConnOffset* or *ClientConnPtr* field in MQCNO contains data that is

not valid. Consult the MQSeries error log for more information about the nature of the error.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that required input fields in the MQCD structure are set correctly.

MQRC_CF_NOT_AVAILABLE
(2345, X'0929')

Explanation: An MQOPEN or MQPUT1 call was issued to access a shared queue, but the allocation of the coupling-facility structure specified in the queue definition failed because there is no suitable coupling facility to hold the structure, based on the preference list in the active CFRM policy.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Make available a coupling facility with one of the names specified in the CFRM policy, or modify the CFRM policy to specify the names of coupling facilities that are available.

MQRC_CF_STRUC_AUTH_FAILED
(2348, X'092C')

Explanation: An MQOPEN or MQPUT1 call was issued to access a shared queue, but the call failed because the user is not authorized to access the coupling-facility structure specified in the queue definition.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Modify the security profile for the user identifier used by the application so that the application can access the coupling-facility structure specified in the queue definition.

MQRC_CF_STRUC_ERROR
(2349, X'092D')

Explanation: An MQOPEN or MQPUT1 call was issued to access a shared queue, but the call failed because the coupling-facility structure name specified in the queue definition is not defined in the CFRM data set, or is not the name of a list structure.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Modify the queue definition to specify the name of a coupling-facility list structure that is defined in the CFRM data set.

MQRC_CF_STRUC_IN_USE
(2346, X'092A')

Explanation: An MQI call was issued to operate on a shared queue, but the call failed because the coupling-facility structure specified in the queue definition is temporarily unavailable. The coupling-facility structure can be unavailable because a structure dump is in progress, or new connectors to the structure are currently inhibited, or an existing connector to the structure failed or disconnected abnormally and clean-up is not yet complete.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: The problem is temporary; wait a short while and then retry the operation.

MQRC_CF_STRUC_LIST_HDR_IN_USE
(2347, X'092B')

Explanation: An MQGET, MQOPEN, MQPUT1, or MQSET call was issued to access a shared queue, but the call failed because the list header associated with the coupling-facility structure specified in the queue definition is temporarily unavailable. The list header is unavailable because it is undergoing recovery processing.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: The problem is temporary; wait a short while and then retry the operation.

MQRC_CFH_ERROR
(2235, X'08BB')

Explanation: On an MQPUT or MQPUT1 call, the PCF header structure MQCFH in the message data is not valid.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Correct the definition of the MQCFH structure. Ensure that the fields are set correctly.

MQRC_CFIL_ERROR
(2236, X'08BC')

Explanation: On an MQPUT or MQPUT1 call, a PCF integer list parameter structure MQCFIL in the message data is not valid.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Correct the definition of the MQCFIL structure. Ensure that the fields are set correctly.

MQRC_CFIN_ERROR
(2237, X'08BD')

Explanation: On an MQPUT or MQPUT1 call, a PCF integer parameter structure MQCFIN in the message data is not valid.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Correct the definition of the MQCFIN structure. Ensure that the fields are set correctly.

MQRC_CFSL_ERROR
(2238, X'08BE')

Explanation: On an MQPUT or MQPUT1 call, a PCF string list parameter structure MQCFSL in the message data is not valid.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Correct the definition of the MQCFSL structure. Ensure that the fields are set correctly.

MQRC_CFST_ERROR
(2239, X'08BF')

Explanation: On an MQPUT or MQPUT1 call, a PCF string parameter structure MQCFST in the message data is not valid.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Correct the definition of the MQCFST structure. Ensure that the fields are set correctly.

MQRC_CHANNEL_ACTIVATED
(2295, X'08F7')

Explanation: This condition is detected when a channel that has been waiting to become active, and for which a Channel Not Activated event has been generated, is now able to become active because an active slot has been released by another channel.

This event is not generated for a channel that is able to become active without waiting for an active slot to be released.

Completion Code: MQCC_WARNING

Programmer Response: None. This reason code is only used to identify the corresponding event message.

MQRC_CHANNEL_AUTO_DEF_ERROR
(2234, X'08BA')

Explanation: This condition is detected when the automatic definition of a channel fails; this may be because an error occurred during the definition process, or because the channel automatic-definition exit inhibited the definition. Additional information is returned in the event message indicating the reason for the failure.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_WARNING

Programmer Response: Examine the additional information returned in the event message to determine the reason for the failure.

MQRC_CHANNEL_AUTO_DEF_OK
(2233, X'08B9')

Explanation: This condition is detected when the automatic definition of a channel is successful. The channel is defined by the MCA.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_WARNING

Programmer Response: None. This reason code is only used to identify the corresponding event message.

MQRC_CHANNEL_CONV_ERROR
(2284, X'08EC')

Explanation: This condition is detected when a channel is unable to do data conversion and the MQGET call to get a message from the transmission queue resulted in a data conversion error. The conversion reason code identifies the reason for the failure.

Completion Code: MQCC_WARNING

Programmer Response: None. This reason code is only used to identify the corresponding event message.

MQRC_CHANNEL_NOT_ACTIVATED (2296, X'08F8')

Explanation: This condition is detected when a channel is required to become active, either because it is starting or because it is about to make another attempt to establish connection with its partner. However, it is unable to do so because the limit on the number of active channels has been reached.

- On OS/390, the maximum number of active channels is given by the ACTCHL parameter in CSQXPARM.
- In other environments, the maximum number of active channels is given by the MaxActiveChannels parameter in the qm.ini file.

The channel waits until it is able to take over an active slot released when another channel ceases to be active. At that time a Channel Activated event is generated.

Completion Code: MQCC_WARNING

Programmer Response: None. This reason code is only used to identify the corresponding event message.

MQRC_CHANNEL_STARTED (2282, X'08EA')

Explanation: One of the following has occurred:

- An operator has issued a Start Channel command.
- An instance of a channel has been successfully established. This condition is detected when Initial Data negotiation is complete and resynchronization has been performed where necessary such that message transfer can proceed.

Completion Code: MQCC_WARNING

Programmer Response: None. This reason code is only used to identify the corresponding event message.

MQRC_CHANNEL_STOPPED (2283, X'08EB')

Explanation: This condition is detected when the channel has been stopped. The reason qualifier identifies the reasons for stopping.

Completion Code: MQCC_WARNING

Programmer Response: None. This reason code is only used to identify the corresponding event message.

MQRC_CHANNEL_STOPPED_BY_USER (2279, X'08E7')

Explanation: This condition is detected when the channel has been stopped by an operator. The reason qualifier identifies the reasons for stopping.

Completion Code: MQCC_WARNING

Programmer Response: None. This reason code is only used to identify the corresponding event message.

MQRC_CHAR_ATTR_LENGTH_ERROR (2006, X'07D6')

Explanation: *CharAttrLength* is negative (for MQINQ or MQSET calls), or is not large enough to hold all selected attributes (MQSET calls only). This reason also occurs if the parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code: MQCC_FAILED

Programmer Response: Specify a value large enough to hold the concatenated strings for all selected attributes.

MQRC_CHAR_ATTRS_ERROR (2007, X'07D7')

Explanation: *CharAttrs* is not valid. The parameter pointer is not valid, or points to read-only storage for MQINQ calls or to storage that is not as long as implied by *CharAttrLength*. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code: MQCC_FAILED

Programmer Response: Correct the parameter.

MQRC_CHAR_ATTRS_TOO_SHORT (2008, X'07D8')

Explanation: For MQINQ calls, *CharAttrLength* is not large enough to contain all of the character attributes for which MQCA_* selectors are specified in the *Selectors* parameter.

The call still completes, with the *CharAttrs* parameter string filled in with as many character attributes as there is room for. Only complete attribute strings are returned: if there is insufficient space remaining to accommodate an attribute in its entirety, that attribute and subsequent character attributes are omitted. Any space at the end of the string not used to hold an attribute is unchanged.

An attribute that represents a set of values (for example, the namelist *Names* attribute) is treated as a single entity—either all of its values are returned, or none.

Completion Code: MQCC_WARNING

Programmer Response: Specify a large enough value, unless only a subset of the values is needed.

MQRC_CHAR_CONVERSION_ERROR (2340, X'0924')

Explanation: This reason code is returned by the Java MQQueueManager constructor when a required character-set conversion is not available. The

| conversion required is between two nonUnicode
| character sets.

| This reason code occurs in the following environment:
| MQSeries Classes for Java on OS/390.

| **Completion Code:** MQCC_FAILED

| **Programmer Response:** Ensure that the National
| Language Resources component of the OS/390
| Language Environment® is installed, and that
| conversion between the IBM-1047 and ISO8859-1
| character sets is available.

MQRC_CICS_BRIDGE_RESTRICTION
(2187, X'088B')

Explanation: It is not permitted to issue MQI calls from user transactions that are run in an MQSeries-CICS bridge environment where the bridge exit also issues MQI calls. The MQI call fails. If this occurs in the bridge exit, it will result in a transaction abend. If it occurs in the user transaction, this may result in a transaction abend.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: The transaction cannot be run using the MQSeries-CICS bridge. Refer to the appropriate CICS manual for information about restrictions in the MQSeries-CICS bridge environment.

MQRC_CICS_WAIT_FAILED
(2140, X'085C')

Explanation: On any MQI call, the CICS adapter issued an EXEC CICS WAIT request, but the request was rejected by CICS.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Examine the CICS trace data for actual response codes. The most likely cause is that the task has been canceled by the operator or by the system.

MQRC_CLIENT_CONN_ERROR
(2278, X'08E6')

Explanation: An MQCONNX call was issued to connect to a queue manager, but the MQCD channel definition structure is not specified correctly. One of the following applies:

- *ClientConnOffset* is not zero and *ClientConnPtr* is not zero and not the null pointer.
- *ClientConnPtr* is not a valid pointer.
- *ClientConnPtr* or *ClientConnOffset* points to storage that is not accessible.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that at least one of *ClientConnOffset* and *ClientConnPtr* is zero. Ensure that the field used points to accessible storage.

MQRC_CLUSTER_EXIT_ERROR
(2266, X'08DA')

Explanation: An MQOPEN, MQPUT, or MQPUT1 call was issued to open or put a message on a cluster queue, but the cluster workload exit defined by the queue-manager's *ClusterWorkloadExit* attribute failed unexpectedly or did not respond in time. Subsequent MQOPEN, MQPUT, and MQPUT1 calls for this queue handle are processed as though the *ClusterWorkloadExit* attribute were blank.

- On OS/390, a message giving more information about the error is written to the system log, for example message CSQV455E or CSQV456E.

This reason code occurs in the following environments: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Check the cluster workload exit to ensure that it has been written correctly.

MQRC_CLUSTER_EXIT_LOAD_ERROR
(2267, X'08DB')

Explanation: An MQCONN or MQCONNX call was issued to connect to a queue manager, but the call failed because the cluster workload exit defined by the queue-manager's *ClusterWorkloadExit* attribute could not be loaded.

- On OS/390, if the cluster workload exit cannot be loaded, a message is written to the system log, for example message CSQV453I. Processing continues as though the *ClusterWorkloadExit* attribute had been blank.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the cluster workload exit has been installed in the correct location.

MQRC_CLUSTER_PUT_INHIBITED
(2268, X'08DC')

Explanation: An MQOPEN call with the MQOO_OUTPUT and MQOO_BIND_ON_OPEN options in effect was issued for a cluster queue, but the

call failed because all of the following are true:
- All instances of the cluster queue are currently put-inhibited (that is, all of the queue instances have the *InhibitPut* attribute set to MQQA_PUT_INHIBITED).
- There is no local instance of the queue. (If there is a local instance, the MQOPEN call succeeds, even if the local instance is put-inhibited.)
- There is no cluster workload exit for the queue, or there is a cluster workload exit but it did not choose a queue instance. (If the cluster workload exit does choose a queue instance, the MQOPEN call succeeds, even if that instance is put-inhibited.)

If the MQOO_BIND_NOT_FIXED option is specified on the MQOPEN call, the call can succeed even if all of the queues in the cluster are put-inhibited. However, a subsequent MQPUT call may fail if all of the queues are still put-inhibited at the time of the MQPUT call.

This reason code occurs in the following environments: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: If the system design allows put requests to be inhibited for short periods, retry the operation later. If the problem persists, determine why all of the queues in the cluster are put-inhibited.

MQRC_CLUSTER_RESOLUTION_ERROR
(2189, X'088D')

Explanation: An MQOPEN, MQPUT, or MQPUT1 call was issued to open or put a message on a cluster queue, but the queue definition could not be resolved correctly because a response was required from the repository manager but none was available.

This reason code occurs in the following environments: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Check that the repository manager is operating and that the queue and channel definitions are correct.

MQRC_CLUSTER_RESOURCE_ERROR
(2269, X'08DD')

Explanation: An MQOPEN, MQPUT, or MQPUT1 call was issued for a cluster queue, but an error occurred whilst trying to use a resource required for clustering.

This reason code occurs in the following environments: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Do the following:
- Check that the SYSTEM.CLUSTER.* queues are not put inhibited or full.
- Check the event queues for any events relating to the SYSTEM.CLUSTER.* queues, as these may give guidance as to the nature of the failure.
- Check that the repository queue manager is available.
- On OS/390, check the console for signs of the failure, such as full page sets.

MQRC_CMD_SERVER_NOT_AVAILABLE
(2322, X'0912')

Explanation: The command server that processes administration commands is not available.

Completion Code: MQCC_FAILED

Programmer Response: Start the command server.

MQRC_CNO_ERROR
(2139, X'085B')

Explanation: On an MQCONNX call, the connect-options structure MQCNO is not valid, for one of the following reasons:
- The *StrucId* field is not MQCNO_STRUC_ID.
- The *Version* field specifies a value that is not valid or not supported.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The queue manager cannot copy the changed structure to application storage, even though the call is successful. This can occur, for example, if the parameter pointer points to read-only storage.

This reason code occurs in the following environments: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Correct the definition of the MQCNO structure. Ensure that required input fields are set correctly.

MQRC_COD_NOT_VALID_FOR_XCF_Q
(2106, X'083A')

Explanation: An MQPUT or MQPUT1 call was issued, but the *Report* field in the message descriptor MQMD specifies one of the MQRO_COD_* options and the target queue is an XCF queue. MQRO_COD_* options cannot be specified for XCF queues.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Remove the relevant MQRO_COD_* option.

MQRC_CODED_CHAR_SET_ID_ERROR
(2330, X'091A')

Explanation: The *CodedCharSetId* parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code: MQCC_FAILED

Programmer Response: Correct the parameter.

MQRC_COMMAND_TYPE_ERROR
(2300, X'08FC')

Explanation: The mqExecute call was issued, but the value of the MQIASY_TYPE data item in the administration bag is not MQCFT_COMMAND.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the MQIASY_TYPE data item in the administration bag has the value MQCFT_COMMAND.

MQRC_CONN_ID_IN_USE
(2160, X'0870')

Explanation: On an MQCONN call, the connection identifier assigned by MQSeries to the connection between a CICS or IMS allied address space and the queue manager conflicts with the connection identifier of another connected CICS or IMS system. The connection identifier assigned is as follows:

- For CICS, the applid
- For IMS, the IMSID parameter on the IMSCTRL (sysgen) macro, or the IMSID parameter on the execution parameter (EXEC card in IMS control region JCL)
- For batch, the job name
- For TSO, the user ID

A conflict arises only if there are two CICS systems, two IMS systems, or one each of CICS and IMS, having the same connection identifiers. Batch and TSO connections need not have unique identifiers.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the naming conventions used in different systems that might connect to MQSeries do not conflict.

MQRC_CONN_TAG_IN_USE
(2271, X'08DF')

Explanation: An MQCONNX call was issued specifying one of the MQCNO_*_CONN_TAG_* options, but the call failed because the connection tag specified by *ConnTag* in MQCNO is in use by an active process or thread, or there is an unresolved unit of work that references this connection tag.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: The problem is likely to be transitory. The application should wait a short while and then retry the operation.

MQRC_CONN_TAG_NOT_RELEASED
(2344, X'0928')

Explanation: An MQDISC call was issued when there was a unit of work outstanding for the connection handle. For CICS, IMS, and RRS connections, the MQDISC call does not commit or back out the unit of work. As a result, the connection tag associated with the unit of work is not yet available for reuse. The tag becomes available for reuse only when processing of the unit of work has been completed.

This reason code occurs only on OS/390.

Completion Code: MQCC_WARNING

Programmer Response: Do not try to reuse the connection tag immediately. If the MQCONNX call is issued with the same connection tag, and that tag is still in use, the call fails with reason code MQRC_CONN_TAG_IN_USE.

MQRC_CONN_TAG_NOT_USABLE
(2350, X'092E')

Explanation: An MQCONNX call was issued specifying one of the MQCNO_*_CONN_TAG_* options, but the call failed because the connection tag specified by *ConnTag* in MQCNO is being used by the queue manager for recovery processing, and this processing is delayed pending recovery of the coupling facility.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: The problem is likely to persist. Consult the system programmer to ascertain the cause of the problem.

MQRC_CONNECTION_BROKEN
(2009, X'07D9')

Explanation: Connection to the queue manager has been lost. This can occur because the queue manager has ended. If the call is an MQGET call with the

MQGMO_WAIT option, the wait has been canceled. All connection and object handles are now invalid.

For MQ client applications, it is possible that the call did complete successfully, even though this reason code is returned with a *CompCode* of MQCC_FAILED.

Completion Code: MQCC_FAILED

Programmer Response: Applications can attempt to reconnect to the queue manager by issuing the MQCONN or MQCONNX call. It may be necessary to poll until a successful response is received.

- On OS/390 for CICS applications, it is not necessary to issue the MQCONN or MQCONNX call, because CICS applications are connected automatically.

Any uncommitted changes in a unit of work should be backed out. A unit of work that is coordinated by the queue manager is backed out automatically.

MQRC_CONNECTION_ERROR (2273, X'08E1')

Explanation: An MQCONN or MQCONNX call failed for one of the following reasons:

- The system parameter module is not at the same release level as the queue manager.
- An internal error was detected by the queue manager.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Relinkedit the system parameter module (CSQZPARM) to ensure that it is at the correct level. If the problem persists, contact your IBM support center.

MQRC_CONNECTION_NOT_AUTHORIZED (2217, X'08A9')

Explanation: This reason code arises only for CICS applications. For these, connection to the queue manager is done by the adapter. If that connection fails because the CICS subsystem is not authorized to connect to the queue manager, this reason code is issued whenever an application running under that subsystem subsequently issues an MQI call.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the subsystem is authorized to connect to the queue manager.

MQRC_CONNECTION_QUIESCING (2202, X'089A')

Explanation: This reason code is issued for CICS and IMS applications when the connection to the queue manager is in quiescing state, and an application issues one of the following calls:

- MQCONN or MQCONNX on IMS (these calls do not return this reason code on CICS, as the calls cannot determine that the queue manager is shutting down)
- MQOPEN, with no connection established, or with MQOO_FAIL_IF_QUIESCING included in the *Options* parameter
- MQGET, with MQGMO_FAIL_IF_QUIESCING included in the *Options* field of the *GetMsgOpts* parameter
- MQPUT or MQPUT1, with MQPMO_FAIL_IF_QUIESCING included in the *Options* field of the *PutMsgOpts* parameter

MQRC_CONNECTION_QUIESCING is also issued by the message channel agent (MCA) when the queue manager is in quiescing state.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: The application should tidy up and terminate.

MQRC_CONNECTION_STOPPING (2203, X'089B')

Explanation: This reason code is issued for CICS and IMS applications when the connection to the queue manager is shutting down, and the application issues an MQI call. No more message-queuing calls can be issued.

- For the MQCONN or MQCONNX call, MQRC_CONNECTION_STOPPING is returned only on IMS. (These calls do not return this reason code on CICS, as the calls cannot determine that the queue manager is shutting down.)
- For the MQGET call, if the MQGMO_WAIT option was specified, the wait is canceled.

Note that the MQRC_CONNECTION_BROKEN reason may be returned instead if, as a result of system scheduling factors, the queue manager shuts down before the call completes.

MQRC_CONNECTION_STOPPING is also issued by the message channel agent (MCA) when the queue manager is shutting down.

For MQ client applications, it is possible that the call did complete successfully, even though this reason code is returned with a *CompCode* of MQCC_FAILED.

This reason code occurs on OS/390, plus MQSeries clients connected to this system.

Completion Code: MQCC_FAILED

Programmer Response: The application should tidy up and terminate. Any uncommitted changes in a unit of work should be backed out. A unit of work that is coordinated by the queue manager is backed out automatically.

MQRC_CONTEXT_HANDLE_ERROR
(2097, X'0831')

Explanation: On an MQPUT or MQPUT1 call, MQPMO_PASS_IDENTITY_CONTEXT or MQPMO_PASS_ALL_CONTEXT was specified, but the handle specified in the *Context* field of the *PutMsgOpts* parameter is either not a valid queue handle, or it is a valid queue handle but the queue was not opened with MQOO_SAVE_ALL_CONTEXT.

Completion Code: MQCC_FAILED

Programmer Response: Specify MQOO_SAVE_ALL_CONTEXT when the queue referred to is opened.

MQRC_CONTEXT_NOT_AVAILABLE
(2098, X'0832')

Explanation: On an MQPUT or MQPUT1 call, MQPMO_PASS_IDENTITY_CONTEXT or MQPMO_PASS_ALL_CONTEXT was specified, but the queue handle specified in the *Context* field of the *PutMsgOpts* parameter has no context associated with it. This arises if no message has yet been successfully retrieved with the queue handle referred to, or if the last successful MQGET call was a browse.

This condition does not arise if the message that was last retrieved had no context associated with it.

- On OS/390, if a message is received by a message channel agent that is putting messages with the authority of the user identifier in the message, this code is returned in the *Feedback* field of an exception report if the message has no context associated with it.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that a successful nonbrowse get call has been issued with the queue handle referred to.

MQRC_CONVERTED_MSG_TOO_BIG
(2120, X'0848')

Explanation: On an MQGET call with the MQGMO_CONVERT option included in the *GetMsgOpts* parameter, the message data expanded during data conversion and exceeded the size of the buffer provided by the application. However, the message had already been removed from the queue because prior to conversion the message data could be accommodated in the application buffer without truncation.

The message is returned unconverted, with the *CompCode* parameter of the MQGET call set to MQCC_WARNING. If the message consists of several parts, each of which is described by its own character-set and encoding fields (for example, a message with format name MQFMT_DEAD_LETTER_HEADER), some parts may be converted and other parts not converted. However, the values returned in the various character-set and encoding fields always correctly describe the relevant message data.

This reason can also occur on the MQXCNVC call, when the *TargetBuffer* parameter is too small too accommodate the converted string, and the string has been truncated to fit in the buffer. The length of valid data returned is given by the *DataLength* parameter; in the case of a DBCS string or mixed SBCS/DBCS string, this length may be *less than* the length of *TargetBuffer*.

Completion Code: MQCC_WARNING

Programmer Response: For the MQGET call, check that the exit is converting the message data correctly and setting the output length *DataLength* to the appropriate value. If it is, the application issuing the MQGET call must provide a larger buffer for the *Buffer* parameter.

For the MQXCNVC call, if the string must be converted without truncation, provide a larger output buffer.

MQRC_CONVERTED_STRING_TOO_BIG
(2190, X'088E')

Explanation: On an MQGET call with the MQGMO_CONVERT option included in the *GetMsgOpts* parameter, a string in a fixed-length field in the message expanded during data conversion and exceeded the size of the field. When this happens, the queue manager tries discarding trailing blank characters and characters following the first null character in order to make the string fit, but in this case there were insufficient characters that could be discarded.

This reason code can also occur for messages with a format name of MQFMT_IMS_VAR_STRING. When this happens, it indicates that the IMS variable string expanded such that its length exceeded the capacity of the 2-byte binary length field contained within the structure of the IMS variable string. (The queue manager never discards trailing blanks in an IMS variable string.)

The message is returned unconverted, with the *CompCode* parameter of the MQGET call set to MQCC_WARNING. If the message consists of several parts, each of which is described by its own character-set and encoding fields (for example, a message with format name MQFMT_DEAD_LETTER_HEADER), some parts may be converted and other parts not converted. However, the values returned in the various character-set and encoding fields always correctly describe the relevant message data.

This reason code does not occur if the string could be made to fit by discarding trailing blank characters.

Completion Code: MQCC_WARNING

Programmer Response: Check that the fields in the message contain the correct values, and that the character-set identifiers specified by the sender and receiver of the message are correct. If they are, the layout of the data in the message must be modified to increase the lengths of the field(s) so that there is sufficient space to allow the string(s) to expand when converted.

MQRC_CORREL_ID_ERROR
(2207, X'089F')

Explanation: An MQGET call was issued to retrieve a message using the correlation identifier as a selection criterion, but the call failed because selection by correlation identifier is not supported on this queue.

- On OS/390, the queue is a shared queue, but the *IndexType* queue attribute does not have an appropriate value:
  - If selection is by correlation identifier alone, *IndexType* must have the value MQIT_CORREL_ID.
  - If selection is by correlation identifier and message identifier combined, *IndexType* must have the value MQIT_CORREL_ID or MQIT_MSG_ID.
- On Tandem NonStop Kernel, a key file is required but has not been defined.

Completion Code: MQCC_FAILED

Programmer Response: Do one of the following:
- On OS/390, change the *IndexType* queue attribute to MQIT_CORREL_ID.
- On Tandem NonStop Kernel, define a key file.
- Modify the application so that it does not use selection by correlation identifier: set the *CorrelId* field to MQCI_NONE and do not specify MQMO_MATCH_CORREL_ID in MQGMO.

MQRC_CURRENT_RECORD_ERROR
(2357, X'0935')

Explanation: An MQXCLWLN call was issued from a cluster workload exit to obtain the address of the next record in the chain, but the address specified by the *CurrentRecord* parameter is not the address of a valid record. *CurrentRecord* must be the address of a destination record (MQWDR), queue record (MQWQR), or cluster record (MQWCR) residing within the cluster cache.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the cluster workload exit passes the address of a valid record residing in the cluster cache.

MQRC_DATA_LENGTH_ERROR
(2010, X'07DA')

Explanation: The *DataLength* parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

This reason can also be returned to an MQ client program that is putting and getting messages, if the application message data is longer than the negotiated maximum message size for the channel.

Completion Code: MQCC_FAILED

Programmer Response: Correct the parameter.

If the error occurs for an MQ client program, also check that the maximum message size for the channel is big enough to accommodate the message being sent; if it is not big enough, increase the maximum message size for the channel.

MQRC_DBCS_ERROR
(2150, X'0866')

Explanation: On the MQXCNVC call, the *SourceCCSID* parameter specifies the coded character-set identifier of a double-byte character set (DBCS), but the *SourceBuffer* parameter does not contain a valid DBCS string. This may be because the string contains characters that are not valid DBCS characters, or because the string is a mixed SBCS/DBCS string and the shift-out/shift-in characters are not correctly paired.

This reason code can also occur on the MQGET call when the MQGMO_CONVERT option is specified. In this case it indicates that the MQRC_DBCS_ERROR reason was returned by an MQXCNVC call issued by the data conversion exit.

Completion Code: MQCC_WARNING or MQCC_FAILED

Programmer Response: Specify a valid string.

If the reason code occurs on the MQGET call, check that the data in the message is valid, and that the logic in the data-conversion exit is correct.

MQRC_DB2_NOT_AVAILABLE
(2342, X'0926')

Explanation: An MQOPEN, MQPUT1, or MQSET call was issued to access a shared queue, but the call failed because the queue manager is not connected to a DB2 subsystem. As a result, the queue manager is unable to access the object definition relating to the shared queue.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Configure the DB2 subsystem so that the queue manager can connect to it.

MQRC_DEF_XMIT_Q_TYPE_ERROR (2198, X'0896')

Explanation: An MQOPEN or MQPUT1 call was issued specifying a remote queue as the destination. Either a local definition of the remote queue was specified, or a queue-manager alias was being resolved, but in either case the *XmitQName* attribute in the local definition is blank.

Because there is no transmission queue defined with the same name as the destination queue manager, the local queue manager has attempted to use the default transmission queue. However, although there is a queue defined by the *DefXmitQName* queue-manager attribute, it is not a local queue.

Completion Code: MQCC_FAILED

Programmer Response: Do one of the following:
- Specify a local transmission queue as the value of the *XmitQName* attribute in the local definition of the remote queue.
- Define a local transmission queue with a name that is the same as that of the remote queue manager.
- Specify a local transmission queue as the value of the *DefXmitQName* queue-manager attribute.

See the *MQSeries Application Programming Guide* for more information.

MQRC_DEF_XMIT_Q_USAGE_ERROR (2199, X'0897')

Explanation: An MQOPEN or MQPUT1 call was issued specifying a remote queue as the destination. Either a local definition of the remote queue was specified, or a queue-manager alias was being resolved, but in either case the *XmitQName* attribute in the local definition is blank.

Because there is no transmission queue defined with the same name as the destination queue manager, the local queue manager has attempted to use the default transmission queue. However, the queue defined by the *DefXmitQName* queue-manager attribute does not have a *Usage* attribute of MQUS_TRANSMISSION.

Completion Code: MQCC_FAILED

Programmer Response: Do one of the following:
- Specify a local transmission queue as the value of the *XmitQName* attribute in the local definition of the remote queue.
- Define a local transmission queue with a name that is the same as that of the remote queue manager.
- Specify a different local transmission queue as the value of the *DefXmitQName* queue-manager attribute.
- Change the *Usage* attribute of the *DefXmitQName* queue to MQUS_TRANSMISSION.

See the *MQSeries Application Programming Guide* for more information.

MQRC_DEST_ENV_ERROR (2263, X'08D7')

Explanation: This reason occurs when a channel exit that processes reference messages detects an error in the destination environment data of a reference message header (MQRMH). One of the following is true:
- *DestEnvLength* is less than zero.
- *DestEnvLength* is greater than zero, but there is no destination environment data.
- *DestEnvLength* is greater than zero, but *DestEnvOffset* is negative, zero, or less than the length of the fixed part of MQRMH.
- *DestEnvLength* is greater than zero, but *DestEnvOffset* plus *DestEnvLength* is greater than *StrucLength*.

The exit returns this reason in the *Feedback* field of the MQCXP structure. If an exception report is requested, it is copied to the *Feedback* field of the MQMD associated with the report.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Specify the destination environment data correctly.

MQRC_DEST_NAME_ERROR (2264, X'08D8')

Explanation: This reason occurs when a channel exit that processes reference messages detects an error in the destination name data of a reference message header (MQRMH). One of the following is true:
- *DestNameLength* is less than zero.
- *DestNameLength* is greater than zero, but there is no destination name data.
- *DestNameLength* is greater than zero, but *DestNameOffset* is negative, zero, or less than the length of the fixed part of MQRMH.
- *DestNameLength* is greater than zero, but *DestNameOffset* plus *DestNameLength* is greater than *StrucLength*.

The exit returns this reason in the *Feedback* field of the MQCXP structure. If an exception report is requested, it is copied to the *Feedback* field of the MQMD associated with the report.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Specify the destination name data correctly.

MQRC_DH_ERROR
(2135, X'0857')

Explanation: On an MQPUT or MQPUT1 call, the distribution header structure MQDH in the message data is not valid.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Correct the definition of the MQDH structure. Ensure that the fields are set correctly.

MQRC_DLH_ERROR
(2141, X'085D')

Explanation: On an MQPUT or MQPUT1 call, the dead letter header structure MQDLH in the message data is not valid.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Correct the definition of the MQDLH structure. Ensure that the fields are set correctly.

MQRC_DUPLICATE_RECOV_COORD
(2163, X'0873')

Explanation: On an MQCONN or MQCONNX call, a recovery coordinator already exists for the connection name specified on the connection call issued by the adapter.

A conflict arises only if there are two CICS systems, two IMS systems, or one each of CICS and IMS, having the same connection identifiers. Batch and TSO connections need not have unique identifiers.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the naming conventions used in different systems that might connect to MQSeries do not conflict.

MQRC_DYNAMIC_Q_NAME_ERROR
(2011, X'07DB')

Explanation: On the MQOPEN call, a model queue is specified in the *ObjectName* field of the *ObjDesc* parameter, but the *DynamicQName* field is not valid, for one of the following reasons:

- *DynamicQName* is completely blank (or blank up to the first null character in the field).
- Characters are present that are not valid for a queue name.
- An asterisk is present beyond the 33rd position (and before any null character).
- An asterisk is present followed by characters that are not null and not blank.

This reason code can also sometimes occur when a server application opens the reply queue specified by the *ReplyToQ* and *ReplyToQMgr* fields in the MQMD of a message that the server has just received. In this case the reason code indicates that the application that sent the original message placed incorrect values into the *ReplyToQ* and *ReplyToQMgr* fields in the MQMD of the original message.

Completion Code: MQCC_FAILED

Programmer Response: Specify a valid name.

MQRC_ENCODING_NOT_SUPPORTED
(2308, X'0904')

Explanation: The *Encoding* field in the message descriptor MQMD contains a value that is not supported:

- For the mqPutBag call, the field in error resides in the *MsgDesc* parameter of the call.
- For the mqGetBag call, the field in error resides in:
  - The *MsgDesc* parameter of the call if the MQGMO_CONVERT option was specified.
  - The message descriptor of the message about to be retrieved if MQGMO_CONVERT was *not* specified.

Completion Code: MQCC_FAILED

Programmer Response: The value must be MQENC_NATIVE.

If the value of the *Encoding* field in the message is not valid, the message cannot be retrieved using the mqGetBag call:

- If one of the MQGMO_BROWSE_* options was specified, the message remains on the queue and can be retrieved using the MQGET call.
- In other cases, the message has already been removed from the queue and discarded. If the message was retrieved within a unit of work, the unit of work can be backed out and the message retrieved using the MQGET call.

MQRC_ENVIRONMENT_ERROR
(2012, X'07DC')

Explanation: The call is not valid for the current environment.

- On OS/390, one of the following applies:
  - An MQCONN or MQCONNX call was issued, but the application has been linked with an adapter that is not supported in the environment in which the application is running. For example, this can arise when the application is linked with the MQ

RRS adapter, but the application is running in a DB2 Stored Procedure address space. RRS is not supported in this environment. Stored Procedures wishing to use the MQ RRS adapter must run in a DB2 WLM-managed Stored Procedure address space.
  - An MQCMIT or MQBACK call was issued in the CICS or IMS environment.
  - The RRS subsystem is not up and running on the OS/390 system that ran the application.
- On Compaq (DIGITAL) OpenVMS, OS/2, AS/400, Tandem NonStop Kernel, UNIX systems, and Windows NT, one of the following applies:
  - The application is linked to the wrong libraries (threaded or nonthreaded).
  - An MQBEGIN, MQCMIT, or MQBACK call was issued, but an external unit-of-work manager is in use. For example, this reason code occurs on Windows NT when an MTS object is running as a DTC transaction. This reason code also occurs if the queue manager does not support units of work.
  - The MQBEGIN call was issued in an MQ client environment.
  - An MQXCLWLN call was issued, but the call did not originate from a cluster workload exit.

Completion Code: MQCC_FAILED

Programmer Response: Do one of the following (as appropriate):

- On OS/390:
  - Link the application with the correct adapter.
  - For a CICS or IMS application, issue the appropriate CICS or IMS call to commit or backout the unit of work.
  - Start the RRS subsystem on the OS/390 system that is running the application.
- In the other environments:
  - Link the application with the correct libraries (threaded or nonthreaded).
  - Remove from the application the call that is not supported.

MQRC_EXPIRY_ERROR
(2013, X'07DD')

Explanation: On an MQPUT or MQPUT1 call, the value specified for the *Expiry* field in the message descriptor MQMD is not valid.

Completion Code: MQCC_FAILED

Programmer Response: Specify a value that is greater than zero, or the special value MQEI_UNLIMITED.

MQRC_FEEDBACK_ERROR
(2014, X'07DE')

Explanation: On an MQPUT or MQPUT1 call, the value specified for the *Feedback* field in the message descriptor MQMD is not valid. The value is not MQFB_NONE, and is outside both the range defined for system feedback codes and the range defined for application feedback codes.

Completion Code: MQCC_FAILED

Programmer Response: Specify MQFB_NONE, or a value in the range MQFB_SYSTEM_FIRST through MQFB_SYSTEM_LAST, or MQFB_APPL_FIRST through MQFB_APPL_LAST.

MQRC_FILE_SYSTEM_ERROR
(2208, X'08A0')

Explanation: An unexpected return code was received from the file system, in attempting to perform an operation on a queue.

This reason code occurs only on VSE/ESA.

Completion Code: MQCC_FAILED

Programmer Response: Check the file system definition for the queue that was being accessed. For a VSAM file, check that the control interval is large enough for the maximum message length allowed for the queue.

MQRC_FORMAT_ERROR
(2110, X'083E')

Explanation: An MQGET call was issued with the MQGMO_CONVERT option specified in the *GetMsgOpts* parameter, but the message cannot be converted successfully due to an error associated with the message format. Possible errors include:

- The format name in the message is MQFMT_NONE.
- A user-written exit with the name specified by the *Format* field in the message cannot be found.
- The message contains data that is not consistent with the format definition.

The message is returned unconverted to the application issuing the MQGET call, the values of the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to those of the message returned, and the call completes with MQCC_WARNING.

If the message consists of several parts, each of which is described by its own *CodedCharSetId* and *Encoding* fields (for example, a message with format name MQFMT_DEAD_LETTER_HEADER), some parts may be converted and other parts not converted. However, the values returned in the various *CodedCharSetId* and *Encoding* fields always correctly describe the relevant message data.

Completion Code: MQCC_WARNING

Programmer Response: Check the format name that was specified when the message was put. If this is not one of the built-in formats, check that a suitable exit with the same name as the format is available for the queue manager to load. Verify that the data in the

message corresponds to the format expected by the exit.

MQRC_FORMAT_NOT_SUPPORTED
(2317, X'090D')

Explanation: The *Format* field in the message descriptor MQMD contains a value that is not supported:
- For the mqPutBag call, the field in error resides in the *MsgDesc* parameter of the call.
- For the mqGetBag call, the field in error resides in the message descriptor of the message about to be retrieved.

Completion Code: MQCC_FAILED

Programmer Response: The value must be one of the following:
MQFMT_ADMIN
MQFMT_EVENT
MQFMT_PCF

If the value of the *Format* field in the message is none of these values, the message cannot be retrieved using the mqGetBag call:
- If one of the MQGMO_BROWSE_* options was specified, the message remains on the queue and can be retrieved using the MQGET call.
- In other cases, the message has already been removed from the queue and discarded. If the message was retrieved within a unit of work, the unit of work can be backed out and the message retrieved using the MQGET call.

MQRC_FUNCTION_ERROR
(2281, X'08E9')

Explanation: The function identifier *Function* specified on the MQZEP call issued by an installable service is not valid for the service being configured.
- On OS/390, this reason code does not occur.

Completion Code: MQCC_FAILED

Programmer Response: Specify an MQZID_* value that is valid for the installable service being configured. Refer to the description of the MQZEP call in the *MQSeries Programmable System Management* book to determine which values are valid.

MQRC_FUNCTION_NOT_SUPPORTED
(2298, X'08FA')

Explanation: The function requested is not available in the current environment.

Completion Code: MQCC_FAILED

Programmer Response: Remove the call from the application.

MQRC_GET_INHIBITED
(2016, X'07E0')

Explanation: MQGET calls are currently inhibited for the queue, or for the queue to which this queue resolves. See the *InhibitGet* queue attribute described in "Chapter 39. Attributes for queues" on page 427.

Completion Code: MQCC_FAILED

Programmer Response: If the system design allows get requests to be inhibited for short periods, retry the operation later.

MQRC_GLOBAL_UOW_CONFLICT
(2351, X'092F')

Explanation: An attempt was made to use inside a global unit of work a connection handle that is participating in another global unit of work. This can occur when an application passes connection handles between objects where the objects are involved in different DTC transactions. Because transaction completion is asynchronous, it is possible for this error to occur *after* the application has finalized the first object and committed its transaction.

This error does not occur for nontransactional MQI calls.

This reason code occurs only on Windows NT.

Completion Code: MQCC_FAILED

Programmer Response: Check that the "MTS Transaction Support" attribute defined for the object's class is set correctly. If necessary, modify the application so that the connection handle is not used by objects participating in different units of work.

MQRC_GMO_ERROR
(2186, X'088A')

Explanation: On an MQGET call, the MQGMO structure is not valid, for one of the following reasons:
- The *StrucId* field is not MQGMO_STRUC_ID.
- The *Version* field specifies a value that is not valid or not supported.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The queue manager cannot copy the changed structure to application storage, even though the call is successful. This can occur, for example, if the pointer points to read-only storage.

Completion Code: MQCC_FAILED

Programmer Response: Correct the definition of the MQGMO structure. Ensure that required input fields are correctly set.

MQRC_GROUP_ID_ERROR
(2258, X'08D2')

Explanation: An MQPUT or MQPUT1 call was issued to put a distribution-list message that is also a message in a group, a message segment, or has segmentation allowed, but an invalid combination of options and values was specified. All of the following are true:

- MQPMO_LOGICAL_ORDER is not specified in the *Options* field in MQPMO.
- Either there are too few MQPMR records provided by MQPMO, or the *GroupId* field is not present in the MQPMR records.
- One or more of the following flags is specified in the *MsgFlags* field in MQMD or MQMDE:
  MQMF_SEGMENTATION_ALLOWED
  MQMF_*_MSG_IN_GROUP
  MQMF_*_SEGMENT
- The *GroupId* field in MQMD or MQMDE is not MQGI_NONE.

This combination of options and values would result in the same group identifier being used for all of the destinations in the distribution list; this is not permitted by the queue manager.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Specify MQGI_NONE for the *GroupId* field in MQMD or MQMDE. Alternatively, if the call is MQPUT specify MQPMO_LOGICAL_ORDER in the *Options* field in MQPMO.

MQRC_HANDLE_IN_USE_FOR_UOW
(2353, X'0931')

Explanation: An attempt was made to use outside a unit of work a connection handle that is participating in a global unit of work.

This error can occur when an application passes connection handles between objects where one object is involved in a DTC transaction and the other is not. Because transaction completion is asynchronous, it is possible for this error to occur *after* the application has finalized the first object and committed its transaction.

This error can also occur when a single object that was created and associated with the transaction loses that association whilst the object is running. The association is lost when DTC terminates the transaction independently of MTS. This might be because the transaction timed out, or because DTC shut down.

This error does not occur for nontransactional MQI calls.

This reason code occurs only on Windows NT.

Completion Code: MQCC_FAILED

Programmer Response: Check that the "MTS Transaction Support" attribute defined for the object's class is set correctly. If necessary, modify the application so that objects executing within different units of work do not try to use the same connection handle.

MQRC_HANDLE_NOT_AVAILABLE
(2017, X'07E1')

Explanation: An MQOPEN or MQPUT1 call was issued, but the maximum number of open handles allowed for the current task has already been reached. Be aware that when a distribution list is specified on the MQOPEN or MQPUT1 call, each queue in the distribution list uses one handle.

- On OS/390, "task" means a CICS task, an MVS task, or an IMS-dependent region.

Completion Code: MQCC_FAILED

Programmer Response: Check whether the application is issuing MQOPEN calls without corresponding MQCLOSE calls. If it is, modify the application to issue the MQCLOSE call for each open object as soon as that object is no longer needed.

Also check whether the application is specifying a distribution list containing a large number of queues that are consuming all of the available handles. If it is, increase the maximum number of handles that the task can use, or reduce the size of the distribution list. The maximum number of open handles that a task can use is given by the *MaxHandles* queue manager attribute (see "Chapter 42. Attributes for the queue manager" on page 469).

MQRC_HBAG_ERROR
(2320, X'0910')

Explanation: A call was issued that has a parameter that is a bag handle, but the handle is not valid. For output parameters, this reason also occurs if the parameter pointer is not valid, or points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code: MQCC_FAILED

Programmer Response: Correct the parameter.

MQRC_HCONFIG_ERROR
(2280, X'08E8')

Explanation: The configuration handle *Hconfig* specified on the MQZEP call issued by an installable service is not valid.

- On OS/390, this reason code does not occur.

Completion Code: MQCC_FAILED

Programmer Response: Specify the configuration handle that was provided to the installable service's configuration function on the component initialization call. See the *MQSeries Programmable System Management* book for information about installable services.

MQRC_HCONN_ERROR
(2018, X'07E2')

Explanation: The connection handle *Hconn* is not valid. This reason also occurs if the parameter pointer is not valid, or (for the MQCONN or MQCONNX call) points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

This reason code can also occur in an MTS environment when trying to use a connection handle in a situation where it is not valid, such as passing it between processes or packages, neither of which is supported. (Passing the connection handle between library packages *is* supported.)

Completion Code: MQCC_FAILED

Programmer Response: Ensure that a successful MQCONN or MQCONNX call is performed for the queue manager, and that an MQDISC call has not already been performed for it. Ensure that the handle is being used within its valid scope (see the MQCONN call described in "Chapter 29. MQCONN - Connect queue manager" on page 333).

- On OS/390, also check that the application has been linked with the correct stub; this is CSQCSTUB for CICS applications, CSQBSTUB for batch applications, and CSQQSTUB for IMS applications. Also, the stub used must not belong to a release of MQSeries that is more recent than the release on which the application will run.

MQRC_HEADER_ERROR
(2142, X'085E')

Explanation: The MQPUT or MQPUT1 call was used to put a message containing an MQ header structure, but the header structure is not valid.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Correct the definition of the MQ header structure. Ensure that the fields are set correctly.

MQRC_HOBJ_ERROR
(2019, X'07E3')

Explanation: The object handle *Hobj* is not valid. This reason also occurs if the parameter pointer is not valid, or (for the MQOPEN call) points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code: MQCC_FAILED

Programmer Response: Ensure that a successful MQOPEN call is performed for this object, and that an MQCLOSE call has not already been performed for it. For MQGET and MQPUT calls, also ensure that the handle represents a queue object. Ensure that the handle is being used within its valid scope (see the MQOPEN call described in "Chapter 34. MQOPEN - Open object" on page 375).

MQRC_IIH_ERROR
(2148, X'0864')

Explanation: On an MQPUT or MQPUT1 call, the IMS information header structure MQIIH in the message data is not valid.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Correct the definition of the MQIIH structure. Ensure that the fields are set correctly.

MQRC_INCOMPLETE_GROUP
(2241, X'08C1')

Explanation: An operation was attempted on a queue using a queue handle that had an incomplete message group. This reason code can arise in the following situations:

- On the MQPUT call, when the application attempts to put a message that is not in a group and specifies MQPMO_LOGICAL_ORDER. The call fails in this case.
- On the MQPUT call, when the application attempts to put a message that is not the next one in the group and does *not* specify MQPMO_LOGICAL_ORDER, but the previous MQPUT call for the queue handle did specify MQPMO_LOGICAL_ORDER. The call succeeds with completion code MQCC_WARNING in this case.
- On the MQGET call, when the application attempts to get a message that is not the next one in the group and does *not* specify MQGMO_LOGICAL_ORDER, but the previous MQGET call for the queue handle did specify MQGMO_LOGICAL_ORDER. The call succeeds with completion code MQCC_WARNING in this case.
- On the MQCLOSE call, when the application attempts to close the queue that has the incomplete message group. The call succeeds with completion code MQCC_WARNING.

If there is an incomplete logical message as well as an incomplete message group, reason code

MQRC_INCOMPLETE_MSG is returned in preference to MQRC_INCOMPLETE_GROUP.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_WARNING or MQCC_FAILED

Programmer Response: If this reason code is expected, no corrective action is required. Otherwise, ensure that the MQPUT call for the last message in the group specifies MQMF_LAST_MSG_IN_GROUP.

MQRC_INCOMPLETE_MSG
(2242, X'08C2')

Explanation: An operation was attempted on a queue using a queue handle that had an incomplete logical message. This reason code can arise in the following situations:

- On the MQPUT call, when the application attempts to put a message that is not a segment and specifies MQPMO_LOGICAL_ORDER. The call fails in this case.
- On the MQPUT call, when the application attempts to put a message that is not the next segment and does *not* specify MQPMO_LOGICAL_ORDER, but the previous MQPUT call for the queue handle did specify MQPMO_LOGICAL_ORDER. The call succeeds with completion code MQCC_WARNING in this case.
- On the MQGET call, when the application attempts to get a message that is not the next segment and does *not* specify MQGMO_LOGICAL_ORDER, but the previous MQGET call for the queue handle did specify MQGMO_LOGICAL_ORDER. The call succeeds with completion code MQCC_WARNING in this case.
- On the MQCLOSE call, when the application attempts to close the queue that has the incomplete logical message. The call succeeds with completion code MQCC_WARNING.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_WARNING or MQCC_FAILED

Programmer Response: If this reason code is expected, no corrective action is required. Otherwise, ensure that the MQPUT call for the last segment specifies MQMF_LAST_SEGMENT.

MQRC_INCONSISTENT_BROWSE
(2259, X'08D3')

Explanation: An MQGET call was issued with the MQGMO_BROWSE_NEXT option specified, but the specification of the MQGMO_LOGICAL_ORDER option for the call is different from the specification of that option for the previous call for the queue handle. Either both calls must specify MQGMO_LOGICAL_ORDER, or neither call must specify MQGMO_LOGICAL_ORDER.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Add or remove the MQGMO_LOGICAL_ORDER option as appropriate. Alternatively, to switch between logical order and physical order, specify the MQGMO_BROWSE_FIRST option to restart the scan from the beginning of the queue, omitting or specifying MQGMO_LOGICAL_ORDER as required.

MQRC_INCONSISTENT_CCSIDS
(2243, X'08C3')

Explanation: An MQGET call was issued specifying the MQGMO_COMPLETE_MSG option, but the message to be retrieved consists of two or more segments that have differing values for the *CodedCharSetId* field in MQMD. This can arise when the segments take different paths through the network, and some of those paths have MCA sender conversion enabled. The call succeeds with a completion code of MQCC_WARNING, but only the first few segments that have identical character-set identifiers are returned.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_WARNING

Programmer Response: Remove the MQGMO_COMPLETE_MSG option from the MQGET call and retrieve the remaining message segments one by one.

MQRC_INCONSISTENT_ENCODINGS
(2244, X'08C4')

Explanation: An MQGET call was issued specifying the MQGMO_COMPLETE_MSG option, but the message to be retrieved consists of two or more segments that have differing values for the *Encoding* field in MQMD. This can arise when the segments take different paths through the network, and some of those paths have MCA sender conversion enabled. The call succeeds with a completion code of MQCC_WARNING, but only the first few segments that have identical encodings are returned.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_WARNING

Programmer Response: Remove the MQGMO_COMPLETE_MSG option from the MQGET call and retrieve the remaining message segments one by one.

MQRC_INCONSISTENT_ITEM_TYPE
(2313, X'0909')

Explanation: The mqAddInteger or mqAddString call was issued to add another occurrence of the specified selector to the bag, but the data type of this occurrence differed from the data type of the first occurrence.

This reason can also occur on the mqBufferToBag and mqGetBag calls, where it indicates that the PCF in the buffer or message contains a selector that occurs more than once but with inconsistent data types.

Completion Code: MQCC_FAILED

Programmer Response: For the mqAddInteger and mqAddString calls, use the call appropriate to the data type of the first occurrence of that selector in the bag.

For the mqBufferToBag and mqGetBag calls, check the logic of the application that created the buffer or sent the message to ensure that multiple-occurrence selectors occur with only one data type. A message that contains a mixture of data types for a selector cannot be retrieved using the mqGetBag call:

- If one of the MQGMO_BROWSE_* options was specified, the message remains on the queue and can be retrieved using the MQGET call.
- In other cases, the message has already been removed from the queue and discarded. If the message was retrieved within a unit of work, the unit of work can be backed out and the message retrieved using the MQGET call.

MQRC_INCONSISTENT_PERSISTENCE
(2185, X'0889')

Explanation: The MQPUT call was issued to put a message that has a value for the *Persistence* field in MQMD that is different from the previous message put using that queue handle. This is not permitted when the MQPMO_LOGICAL_ORDER option is specified and there is already a current message group or logical message. All messages in a group and all segments in a logical message must be persistent, or all must be nonpersistent.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Modify the application to ensure that all of the messages in the group or logical message are put with the same value for the *Persistence* field in MQMD.

MQRC_INCONSISTENT_UOW
(2245, X'08C5')

Explanation: One of the following applies:

- An MQPUT call was issued to put a message in a group or a segment of a logical message, but the value specified or defaulted for the MQPMO_SYNCPOINT option is not consistent with the current group and segment information retained by the queue manager for the queue handle.

  If the current call specifies MQPMO_LOGICAL_ORDER, the call fails. If the current call does not specify MQPMO_LOGICAL_ORDER, but the previous MQPUT call for the queue handle did, the call succeeds with completion code MQCC_WARNING.
- An MQGET call was issued to remove from the queue a message in a group or a segment of a logical message, but the value specified or defaulted for the MQGMO_SYNCPOINT option is not consistent with the current group and segment information retained by the queue manager for the queue handle.

  If the current call specifies MQGMO_LOGICAL_ORDER, the call fails. If the current call does not specify MQGMO_LOGICAL_ORDER, but the previous MQGET call for the queue handle did, the call succeeds with completion code MQCC_WARNING.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_WARNING or MQCC_FAILED

Programmer Response: Modify the application to ensure that the same unit-of-work specification is used for all messages in the group, or all segments of the logical message.

MQRC_INDEX_ERROR
(2314, X'090A')

Explanation: An index parameter to a call or method has a value that is not valid. The value must be zero or greater. For bag calls, certain MQIND_* values can also be specified:

- For the mqDeleteItem, mqSetInteger and mqSetString calls, MQIND_ALL and MQIND_NONE are valid.
- For the mqInquireBag, mqInquireInteger, mqInquireString, and mqInquireItemInfo calls, MQIND_NONE is valid.

Completion Code: MQCC_FAILED

Programmer Response: Specify a valid value.

MQRC_INDEX_NOT_PRESENT
(2306, X'0902')

Explanation: The specified index is not present:
- For a bag, this means that the bag contains one or more data items that have the selector value specified by the *Selector* parameter, but none of them has the index value specified by the *ItemIndex* parameter. The data item identified by the *Selector* and *ItemIndex* parameters must exist in the bag.
- For a namelist, this means that the index parameter value is too large, and outside the range of valid values.

Completion Code: MQCC_FAILED

Programmer Response: Specify the index of a data item that does exist in the bag or namelist. Use the mqCountItems call to determine the number of data items with the specified selector that exist in the bag, or the nameCount method to determine the number of names in the namelist.

MQRC_INHIBIT_VALUE_ERROR
(2020, X'07E4')

Explanation: On an MQSET call, the value specified for either the MQIA_INHIBIT_GET attribute or the MQIA_INHIBIT_PUT attribute is not valid.

Completion Code: MQCC_FAILED

Programmer Response: Specify a valid value. See the *InhibitGet* or *InhibitPut* attribute described in "Chapter 39. Attributes for queues" on page 427.

MQRC_INITIALIZATION_FAILED
(2286, X'08EE')

Explanation: This reason should be returned by an installable service component when the component is unable to complete initialization successfully.
- On OS/390, this reason code does not occur.

Completion Code: MQCC_FAILED

Programmer Response: Correct the error and retry the operation.

MQRC_INQUIRY_COMMAND_ERROR
(2324, X'0914')

Explanation: The mqAddInquiry call was used previously to add attribute selectors to the bag, but the command code to be used for the mqBagToBuffer, mqExecute, or mqPutBag call is not recognized. As a result, the correct PCF message cannot be generated.

Completion Code: MQCC_FAILED

Programmer Response: Remove the mqAddInquiry calls and use instead the mqAddInteger call with the appropriate MQIACF_*_ATTRS or MQIACH_*_ATTRS selectors.

MQRC_INT_ATTR_COUNT_ERROR
(2021, X'07E5')

Explanation: On an MQINQ or MQSET call, the *IntAttrCount* parameter is negative (MQINQ or MQSET), or smaller than the number of integer attribute selectors (MQIA_*) specified in the *Selectors* parameter (MQSET only). This reason also occurs if the parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code: MQCC_FAILED

Programmer Response: Specify a value large enough for all selected integer attributes.

MQRC_INT_ATTR_COUNT_TOO_SMALL
(2022, X'07E6')

Explanation: On an MQINQ call, the *IntAttrCount* parameter is smaller than the number of integer attribute selectors (MQIA_*) specified in the *Selectors* parameter.

The call completes with MQCC_WARNING, with the *IntAttrs* array filled in with as many integer attributes as there is room for.

Completion Code: MQCC_WARNING

Programmer Response: Specify a large enough value, unless only a subset of the values is needed.

MQRC_INT_ATTRS_ARRAY_ERROR
(2023, X'07E7')

Explanation: On an MQINQ or MQSET call, the *IntAttrs* parameter is not valid. The parameter pointer is not valid (MQINQ and MQSET), or points to read-only storage or to storage that is not as long as indicated by the *IntAttrCount* parameter (MQINQ only). (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code: MQCC_FAILED

Programmer Response: Correct the parameter.

MQRC_INVALID_MSG_UNDER_CURSOR
(2246, X'08C6')

Explanation: An MQGET call was issued specifying the MQGMO_COMPLETE_MSG option with either MQGMO_MSG_UNDER_CURSOR or MQGMO_BROWSE_MSG_UNDER_CURSOR, but the message that is under the cursor has an MQMD with an *Offset* field that is greater than zero. Because MQGMO_COMPLETE_MSG was specified, the message is not valid for retrieval.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT,

plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Reposition the browse cursor so that it is located on a message whose *Offset* field in MQMD is zero. Alternatively, remove the MQGMO_COMPLETE_MSG option.

MQRC_ITEM_COUNT_ERROR
(2316, X'090C')

Explanation: The mqTruncateBag call was issued, but the *ItemCount* parameter specifies a value that is not valid. The value is either less than zero, or greater than the number of user-defined data items in the bag.

This reason also occurs on the mqCountItems call if the parameter pointer is not valid, or points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code: MQCC_FAILED

Programmer Response: Specify a valid value. Use the mqCountItems call to determine the number of user-defined data items in the bag.

MQRC_ITEM_TYPE_ERROR
(2327, X'0917')

Explanation: The mqInquireItemInfo call was issued, but the *ItemType* parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code: MQCC_FAILED

Programmer Response: Correct the parameter.

MQRC_ITEM_VALUE_ERROR
(2319, X'090F')

Explanation: The mqInquireBag or mqInquireInteger call was issued, but the *ItemValue* parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code: MQCC_FAILED

Programmer Response: Correct the parameter.

MQRC_LOCAL_UOW_CONFLICT
(2352, X'0930')

Explanation: An attempt was made to use inside a global unit of work a connection handle that is participating in a queue-manager coordinated local unit of work. This can occur when an application passes connection handles between objects where one object is involved in a DTC transaction and the other is not.

This error does not occur for nontransactional MQI calls.

This reason code occurs only on Windows NT.

Completion Code: MQCC_FAILED

Programmer Response: Check that the "MTS Transaction Support" attribute defined for the object's class is set correctly. If necessary, modify the application so that the connection handle is not used by objects participating in different units of work.

MQRC_MATCH_OPTIONS_ERROR
(2247, X'08C7')

Explanation: An MQGET call was issued, but the value of the *MatchOptions* field in the *GetMsgOpts* parameter is not valid. Either an undefined option is specified, or a defined option that is not valid in the current circumstances is specified. In the latter case, it means that all of the following are true:

- MQGMO_LOGICAL_ORDER is specified.
- There is a current message group or logical message for the queue handle.
- Neither of the following options is specified:
  MQGMO_BROWSE_MSG_UNDER_CURSOR
  MQGMO_MSG_UNDER_CURSOR
- One or more of the MQMO_* options is specified.
- The values of the fields in the *MsgDesc* parameter corresponding to the MQMO_* options specified, differ from the values of those fields in the MQMD for the message to be returned next.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that only valid options are specified for the field.

MQRC_MAX_CONNS_LIMIT_REACHED
(2025, X'07E9')

Explanation: The MQCONN or MQCONNX call was rejected because the maximum number of concurrent connections has been exceeded.

- On OS/390, connection limits are applicable only to TSO and batch requests. The limits are determined by the customer using the following parameters of the CSQ6SYSP macro:
  – For TSO: IDFORE
  – For batch: IDBACK

  For more information, see the *MQSeries for OS/390 System Setup Guide*.

- On Compaq (DIGITAL) OpenVMS, OS/2, AS/400, Tandem NonStop Kernel, UNIX systems, and Windows NT, this reason code can also occur on the MQOPEN call.

Completion Code: MQCC_FAILED

Programmer Response: Either increase the size of the appropriate install parameter value, or reduce the number of concurrent connections.

MQRC_MD_ERROR
(2026, X'07EA')

Explanation: The MQMD structure is not valid, for one of the following reasons:
- The *StrucId* field is not MQMD_STRUC_ID.
- The *Version* field specifies a value that is not valid or not supported.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The queue manager cannot copy the changed structure to application storage, even though the call is successful. This can occur, for example, if the pointer points to read-only storage.

Completion Code: MQCC_FAILED

Programmer Response: Correct the definition of the message descriptor. Ensure that required input fields are correctly set.

MQRC_MDE_ERROR
(2248, X'08C8')

Explanation: The MQMDE structure at the start of the application message data is not valid, for one of the following reasons:
- The *StrucId* field is not MQMDE_STRUC_ID.
- The *Version* field is less than MQMDE_VERSION_2.
- The *StrucLength* field is less than MQMDE_LENGTH_2, or (for *Version* equal to MQMDE_VERSION_2 only) greater than MQMDE_LENGTH_2.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Correct the definition of the message descriptor extension. Ensure that required input fields are correctly set.

MQRC_MISSING_REPLY_TO_Q
(2027, X'07EB')

Explanation: On an MQPUT or MQPUT1 call, the *ReplyToQ* field in the message descriptor MQMD is blank, but one or both of the following is true:
- A reply was requested (that is, MQMT_REQUEST was specified in the *MsgType* field of the message descriptor).
- A report message was requested in the *Report* field of the message descriptor.

Completion Code: MQCC_FAILED

Programmer Response: Specify the name of the queue to which the reply message or report message is to be sent.

MQRC_MISSING_WIH
(2332, X'091C')

Explanation: An MQPUT or MQPUT1 call was issued to put a message on a queue whose *IndexType* attribute had the value MQIT_MSG_TOKEN, but the *Format* field in the MQMD was not MQFMT_WORK_INFO_HEADER. This error occurs only when the message arrives at the destination queue manager.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Modify the application to ensure that it places an MQWIH structure at the start of the message data, and sets the *Format* field in the MQMD to MQFMT_WORK_INFO_HEADER.

MQRC_MSG_FLAGS_ERROR
(2249, X'08C9')

Explanation: An MQPUT or MQPUT1 call was issued, but the *MsgFlags* field in the message descriptor MQMD contains one or more message flags that are not recognized by the local queue manager. The message flags that cause this reason code to be returned depend on the destination of the message; see "Appendix E. Report options and message flags" on page 593 for more details.

This reason code can also occur in the *Feedback* field in the MQMD of a report message, or in the *Reason* field in the MQDLH structure of a message on the dead-letter queue; in both cases it indicates that the destination queue manager does not support one or more of the message flags specified by the sender of the message.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Do the following:
- Ensure that the *MsgFlags* field in the message descriptor is initialized with a value when the message descriptor is declared, or is assigned a value prior to the MQPUT or MQPUT1 call. Specify MQMF_NONE if no message flags are needed.

- Ensure that the message flags specified are ones that are documented in this book; see the *MsgFlags* field described in "Chapter 9. MQMD - Message descriptor" on page 125 for valid message flags. Remove any message flags that are not documented in this book.
- If multiple message flags are being set by adding the individual message flags together, ensure that the same message flag is not added twice.

MQRC_MSG_ID_ERROR
(2206, X'089E')

Explanation: An MQGET call was issued to retrieve a message using the message identifier as a selection criterion, but the call failed because selection by message identifier is not supported on this queue.

- On OS/390, the queue is a shared queue, but the *IndexType* queue attribute does not have an appropriate value:
  - If selection is by message identifier alone, *IndexType* must have the value MQIT_MSG_ID.
  - If selection is by message identifier and correlation identifier combined, *IndexType* must have the value MQIT_MSG_ID or MQIT_CORREL_ID.
- On Tandem NonStop Kernel, a key file is required but has not been defined.

Completion Code: MQCC_FAILED

Programmer Response: Do one of the following:
- Modify the application so that it does not use selection by message identifier: set the *MsgId* field to MQMI_NONE and do not specify MQMO_MATCH_MSG_ID in MQGMO.
- On OS/390, change the *IndexType* queue attribute to MQIT_MSG_ID.
- On Tandem NonStop Kernel, define a key file.

MQRC_MSG_SEQ_NUMBER_ERROR
(2250, X'08CA')

Explanation: An MQGET, MQPUT, or MQPUT1 call was issued, but the value of the *MsgSeqNumber* field in the MQMD or MQMDE structure is less than one or greater than 999 999 999.

This error can also occur on the MQPUT call if the *MsgSeqNumber* field would have become greater than 999 999 999 as a result of the call.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Specify a value in the range 1 through 999 999 999. Do not attempt to create a message group containing more than 999 999 999 messages.

MQRC_MSG_TOKEN_ERROR
(2331, X'091B')

Explanation: An MQGET call was issued to retrieve a message using the message token as a selection criterion, but the options specified are not valid, for one of the following reasons:
- MQMO_MATCH_MSG_TOKEN was specified, but the queue is not indexed by message token (that is, the queue's *IndexType* attribute does not have the value MQIT_MSG_TOKEN).
- MQMO_MATCH_MSG_TOKEN was specified with either MQGMO_WAIT or MQGMO_SET_SIGNAL.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Do one of the following:
- Modify the queue definition so that the queue is indexed by message token.
- Remove the MQMO_MATCH_MSG_TOKEN option from the MQGET call.

MQRC_MSG_TOO_BIG_FOR_CHANNEL
(2218, X'08AA')

Explanation: A message was put to a remote queue, but the message is larger than the maximum message length allowed by the channel. This reason code is returned in the *Feedback* field in the message descriptor of a report message.

- On OS/390, this return code is issued only if you are not using CICS for distributed queuing. Otherwise, MQRC_MSG_TOO_BIG_FOR_Q_MGR is issued.

Completion Code: MQCC_FAILED

Programmer Response: Check the channel definitions. Increase the maximum message length that the channel can accept, or break the message into several smaller messages.

MQRC_MSG_TOO_BIG_FOR_Q
(2030, X'07EE')

Explanation: An MQPUT or MQPUT1 call was issued to put a message on a queue, but the message was too long for the queue and MQMF_SEGMENTATION_ALLOWED was not specified in the *MsgFlags* field in MQMD. If segmentation is not allowed, the length of the message cannot exceed the lesser of the queue and queue-manager *MaxMsgLength* attributes.

This reason code can also occur when MQMF_SEGMENTATION_ALLOWED *is* specified, but the nature of the data present in the message prevents the queue manager splitting it into segments that are small enough to place on the queue:
- For a user-defined format, the smallest segment that the queue manager can create is 16 bytes.

- For a built-in format, the smallest segment that the queue manager can create depends on the particular format, but is greater than 16 bytes in all cases other than MQFMT_STRING (for MQFMT_STRING the minimum segment size is 16 bytes).

MQRC_MSG_TOO_BIG_FOR_Q can also occur in the *Feedback* field in the message descriptor of a report message; in this case it indicates that the error was encountered by a message channel agent when it attempted to put the message on a remote queue.

Completion Code: MQCC_FAILED

Programmer Response: Check whether the *BufferLength* parameter is specified correctly; if it is, do one of the following:
- Increase the value of the queue's *MaxMsgLength* attribute; the queue-manager's *MaxMsgLength* attribute may also need increasing.
- Break the message into several smaller messages.
- Specify MQMF_SEGMENTATION_ALLOWED in the *MsgFlags* field in MQMD; this will allow the queue manager to break the message into segments.

MQRC_MSG_TOO_BIG_FOR_Q_MGR
(2031, X'07EF')

Explanation: An MQPUT or MQPUT1 call was issued to put a message on a queue, but the message was too long for the queue manager and MQMF_SEGMENTATION_ALLOWED was not specified in the *MsgFlags* field in MQMD. If segmentation is not allowed, the length of the message cannot exceed the lesser of the queue and queue-manager *MaxMsgLength* attributes.

This reason code can also occur when MQMF_SEGMENTATION_ALLOWED *is* specified, but the nature of the data present in the message prevents the queue manager splitting it into segments that are small enough for the queue-manager limit:
- For a user-defined format, the smallest segment that the queue manager can create is 16 bytes.
- For a built-in format, the smallest segment that the queue manager can create depends on the particular format, but is greater than 16 bytes in all cases other than MQFMT_STRING (for MQFMT_STRING the minimum segment size is 16 bytes).

MQRC_MSG_TOO_BIG_FOR_Q_MGR can also occur in the *Feedback* field in the message descriptor of a report message; in this case it indicates that the error was encountered by a message channel agent when it attempted to put the message on a remote queue.

This reason also occurs if a channel, through which the message is to pass, has restricted the maximum message length to a value that is actually less than that supported by the queue manager, and the message length is greater than this value.
- On OS/390, this return code is issued only if you are using CICS for distributed queuing. Otherwise, MQRC_MSG_TOO_BIG_FOR_CHANNEL is issued.

Completion Code: MQCC_FAILED

Programmer Response: Check whether the *BufferLength* parameter is specified correctly; if it is, do one of the following:
- Increase the value of the queue-manager's *MaxMsgLength* attribute; the queue's *MaxMsgLength* attribute may also need increasing.
- Break the message into several smaller messages.
- Specify MQMF_SEGMENTATION_ALLOWED in the *MsgFlags* field in MQMD; this will allow the queue manager to break the message into segments.
- Check the channel definitions.

MQRC_MSG_TYPE_ERROR
(2029, X'07ED')

Explanation: On an MQPUT or MQPUT1 call, the value specified for the *MsgType* field in the message descriptor (MQMD) is not valid.

Completion Code: MQCC_FAILED

Programmer Response: Specify a valid value. See the *MsgType* field described in "Chapter 9. MQMD - Message descriptor" on page 125 for details.

MQRC_MULTIPLE_INSTANCE_ERROR
(2301, X'08FD')

Explanation: The *Selector* parameter specifies a system selector (one of the MQIASY_* values), but the value of the *ItemIndex* parameter is not MQIND_NONE. Only one instance of each system selector can exist in the bag.

Completion Code: MQCC_FAILED

Programmer Response: Specify MQIND_NONE for the *ItemIndex* parameter.

MQRC_MULTIPLE_REASONS
(2136, X'0858')

Explanation: An MQOPEN, MQPUT or MQPUT1 call was issued to open a distribution list or put a message to a distribution list, but the result of the call was not the same for all of the destinations in the list. One of the following applies:
- The call succeeded for some of the destinations but not others. The completion code is MQCC_WARNING in this case.
- The call failed for all of the destinations, but for differing reasons. The completion code is MQCC_FAILED in this case.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_WARNING or MQCC_FAILED

Programmer Response: Examine the MQRR response records to identify the destinations for which the call failed, and the reason for the failure. Ensure that sufficient response records are provided by the application on the call to enable the error(s) to be determined. For the MQPUT1 call, the response records must be specified using the MQOD structure, and not the MQPMO structure.

MQRC_NAME_IN_USE
(2201, X'0899')

Explanation: An MQOPEN call was issued to create a dynamic queue, but a queue with the same name as the dynamic queue already exists. The existing queue is one that is logically deleted, but for which there are still one or more open handles. For more information, see "Chapter 27. MQCLOSE - Close object" on page 319.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Either ensure that all handles for the previous dynamic queue are closed, or ensure that the name of the new queue is unique; see the description for reason code MQRC_OBJECT_ALREADY_EXISTS.

MQRC_NAME_NOT_VALID_FOR_TYPE
(2194, X'0892')

Explanation: An MQOPEN call was issued to open the queue manager definition, but the *ObjectName* field in the *ObjDesc* parameter is not blank.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the *ObjectName* field is set to blanks.

MQRC_NESTED_BAG_NOT_SUPPORTED
(2325, X'0915')

Explanation: A bag that is input to the call contains nested bags. Nested bags are supported only for bags that are output from the call.

Completion Code: MQCC_FAILED

Programmer Response: Use a different bag as input to the call.

MQRC_NEXT_OFFSET_ERROR
(2358, X'0936')

Explanation: An MQXCLWLN call was issued from a cluster workload exit to obtain the address of the next record in the chain, but the offset specified by the *NextOffset* parameter is not valid. *NextOffset* must be the value of one of the following fields:

- *ChannelDefOffset* field in MQWDR
- *ClusterRecOffset* field in MQWDR
- *ClusterRecOffset* field in MQWQR
- *ClusterRecOffset* field in MQWCR

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the value specified for the *NextOffset* parameter is the value of one of the fields listed above.

MQRC_NO_DESTINATIONS_AVAILABLE
(2270, X'08DE')

Explanation: An MQPUT or MQPUT1 call was issued to put a message on a cluster queue, but at the time of the call there were no longer any instances of the queue in the cluster. The message therefore could not be sent.

This situation can occur when MQOO_BIND_NOT_FIXED is specified on the MQOPEN call that opens the queue, or MQPUT1 is used to put the message.

This reason code occurs in the following environments: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Check the queue definition and queue status to determine why all instances of the queue were removed from the cluster. Correct the problem and rerun the application.

MQRC_NO_EXTERNAL_PARTICIPANTS
(2121, X'0849')

Explanation: An MQBEGIN call was issued to start a unit of work coordinated by the queue manager, but no participating resource managers have been registered with the queue manager. As a result, only changes to MQ resources can be coordinated by the queue manager in the unit of work.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT.

Completion Code: MQCC_WARNING

Programmer Response: If the application does not require non-MQ resources to participate in the unit of work, this reason code can be ignored or the MQBEGIN call removed. Otherwise consult your system support programmer to determine why the required resource managers have not been registered with the queue manager; the queue manager's configuration file may be in error.

MQRC_NO_MSG_AVAILABLE
(2033, X'07F1')

Explanation: An MQGET call was issued, but there is no message on the queue satisfying the selection criteria specified in MQMD (the *MsgId* and *CorrelId* fields), and in MQGMO (the *Options* and *MatchOptions* fields). Either the MQGMO_WAIT option was not specified, or the time interval specified by the *WaitInterval* field in MQGMO has expired. This reason is also returned for an MQGET call for browse, when the end of the queue has been reached.

This reason code can also be returned by the mqGetBag and mqExecute calls. mqGetBag is similar to MQGET. For the mqExecute call, the completion code can be either MQCC_WARNING or MQCC_FAILED:

- If the completion code is MQCC_WARNING, some response messages were received during the specified wait interval, but not all. The response bag contains system-generated nested bags for the messages that were received.
- If the completion code is MQCC_FAILED, no response messages were received during the specified wait interval.

Completion Code: MQCC_WARNING or MQCC_FAILED

Programmer Response: If this is an expected condition, no corrective action is required.

If this is an unexpected condition, check that:

- The message was put on the queue successfully.
- The unit of work (if any) used for the MQPUT or MQPUT1 call was committed successfully.
- The options controlling the selection criteria are specified correctly. All of the following can affect the eligibility of a message for return on the MQGET call:
  MQGMO_LOGICAL_ORDER
  MQGMO_ALL_MSGS_AVAILABLE
  MQGMO_ALL_SEGMENTS_AVAILABLE
  MQGMO_COMPLETE_MSG
  MQMO_MATCH_MSG_ID
  MQMO_MATCH_CORREL_ID
  MQMO_MATCH_GROUP_ID
  MQMO_MATCH_MSG_SEQ_NUMBER
  MQMO_MATCH_OFFSET
  Value of *MsgId* field in MQMD
  Value of *CorrelId* field in MQMD

Consider waiting longer for the message.

MQRC_NO_MSG_LOCKED
(2209, X'08A1')

Explanation: An MQGET call was issued with the MQGMO_UNLOCK option, but no message was currently locked.

- On OS/390, this reason code does not occur.

Completion Code: MQCC_WARNING

Programmer Response: Check that a message was locked by an earlier MQGET call with the MQGMO_LOCK option for the same handle, and that no intervening call has caused the message to become unlocked.

MQRC_NO_MSG_UNDER_CURSOR
(2034, X'07F2')

Explanation: An MQGET call was issued with either the MQGMO_MSG_UNDER_CURSOR or the MQGMO_BROWSE_MSG_UNDER_CURSOR option. However, the browse cursor is not positioned at a retrievable message. This is caused by one of the following:

- The cursor is positioned logically before the first message (as it is before the first MQGET call with a browse option has been successfully performed).
- The message the browse cursor was positioned on has been locked or removed from the queue (probably by some other application) since the browse operation was performed.
- The message the browse cursor was positioned on has expired.

Completion Code: MQCC_FAILED

Programmer Response: Check the application logic. This may be an expected reason if the application design allows multiple servers to compete for messages after browsing. Consider also using the MQGMO_LOCK option with the preceding browse MQGET call.

MQRC_NO_RECORD_AVAILABLE
(2359, X'0937')

Explanation: An MQXCLWLN call was issued from a cluster workload exit to obtain the address of the next record in the chain, but the current record is the last record in the chain.

Completion Code: MQCC_FAILED

Programmer Response: None.

MQRC_NONE
(0, X'0000')

Explanation: The call completed normally. The completion code (*CompCode*) is MQCC_OK.

Completion Code: MQCC_OK

Programmer Response: None.

MQRC_NOT_AUTHORIZED
(2035, X'07F3')

Explanation: The user is not authorized to perform the operation attempted:

- On an MQCONN or MQCONNX call, the user is not authorized to connect to the queue manager.

  - On OS/390, for CICS applications, MQRC_CONNECTION_NOT_AUTHORIZED is issued instead.
- On an MQOPEN or MQPUT1 call, the user is not authorized to open the object for the option(s) specified.
  - On OS/390, if the object being opened is a model queue, this reason also arises if the user is not authorized to create a dynamic queue with the required name.
- On an MQCLOSE call, the user is not authorized to delete the object, which is a permanent dynamic queue, and the *Hobj* parameter specified on the MQCLOSE call is not the handle returned by the MQOPEN call that created the queue.

This reason code can also occur in the *Feedback* field in the message descriptor of a report message; in this case it indicates that the error was encountered by a message channel agent when it attempted to put the message on a remote queue.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the correct queue manager or object was specified, and that appropriate authority exists.

- On OS/390, to determine for which object you are not authorized, you can use the violation messages issued by the External Security Manager.

MQRC_NOT_CONVERTED
(2119, X'0847')

Explanation: An MQGET call was issued with the MQGMO_CONVERT option specified in the *GetMsgOpts* parameter, but an error occurred during conversion of the data in the message. The message data is returned unconverted, the values of the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to those of the message returned, and the call completes with MQCC_WARNING.

If the message consists of several parts, each of which is described by its own *CodedCharSetId* and *Encoding* fields (for example, a message with format name MQFMT_DEAD_LETTER_HEADER), some parts may be converted and other parts not converted. However, the values returned in the various *CodedCharSetId* and *Encoding* fields always correctly describe the relevant message data.

This error may also indicate that a parameter to the data-conversion service is not supported.

Completion Code: MQCC_WARNING

Programmer Response: Check that the message data is correctly described by the *Format*, *CodedCharSetId* and *Encoding* parameters that were specified when the message was put. Also check that these values, and the *CodedCharSetId* and *Encoding* specified in the *MsgDesc* parameter on the MQGET call, are supported for queue-manager conversion. If the required conversion is not supported, conversion must be carried out by the application.

MQRC_NOT_OPEN_FOR_BROWSE
(2036, X'07F4')

Explanation: An MQGET call was issued with one of the following options:
MQGMO_BROWSE_FIRST
MQGMO_BROWSE_NEXT
MQGMO_BROWSE_MSG_UNDER_CURSOR
MQGMO_MSG_UNDER_CURSOR

but the queue had not been opened for browse.

Completion Code: MQCC_FAILED

Programmer Response: Specify MQOO_BROWSE when the queue is opened.

MQRC_NOT_OPEN_FOR_INPUT
(2037, X'07F5')

Explanation: An MQGET call was issued to retrieve a message from a queue, but the queue had not been opened for input.

Completion Code: MQCC_FAILED

Programmer Response: Specify one of the following when the queue is opened:
MQOO_INPUT_SHARED
MQOO_INPUT_EXCLUSIVE
MQOO_INPUT_AS_Q_DEF

MQRC_NOT_OPEN_FOR_INQUIRE
(2038, X'07F6')

Explanation: An MQINQ call was issued to inquire object attributes, but the object had not been opened for inquire.

Completion Code: MQCC_FAILED

Programmer Response: Specify MQOO_INQUIRE when the object is opened.

MQRC_NOT_OPEN_FOR_OUTPUT
(2039, X'07F7')

Explanation: An MQPUT call was issued to put a message on a queue, but the queue had not been opened for output.

Completion Code: MQCC_FAILED

Programmer Response: Specify MQOO_OUTPUT when the queue is opened.

MQRC_NOT_OPEN_FOR_PASS_ALL
(2093, X'082D')

Explanation: An MQPUT call was issued with the MQPMO_PASS_ALL_CONTEXT option specified in the *PutMsgOpts* parameter, but the queue had not been opened with the MQOO_PASS_ALL_CONTEXT option.

Completion Code: MQCC_FAILED

Programmer Response: Specify MQOO_PASS_ALL_CONTEXT (or another option that implies it) when the queue is opened.

MQRC_NOT_OPEN_FOR_PASS_IDENT
(2094, X'082E')

Explanation: An MQPUT call was issued with the MQPMO_PASS_IDENTITY_CONTEXT option specified in the *PutMsgOpts* parameter, but the queue had not been opened with the MQOO_PASS_IDENTITY_CONTEXT option.

Completion Code: MQCC_FAILED

Programmer Response: Specify MQOO_PASS_IDENTITY_CONTEXT (or another option that implies it) when the queue is opened.

MQRC_NOT_OPEN_FOR_SET
(2040, X'07F8')

Explanation: An MQSET call was issued to set queue attributes, but the queue had not been opened for set.

Completion Code: MQCC_FAILED

Programmer Response: Specify MQOO_SET when the object is opened.

MQRC_NOT_OPEN_FOR_SET_ALL
(2095, X'082F')

Explanation: An MQPUT call was issued with the MQPMO_SET_ALL_CONTEXT option specified in the *PutMsgOpts* parameter, but the queue had not been opened with the MQOO_SET_ALL_CONTEXT option.

Completion Code: MQCC_FAILED

Programmer Response: Specify MQOO_SET_ALL_CONTEXT when the queue is opened.

MQRC_NOT_OPEN_FOR_SET_IDENT
(2096, X'0830')

Explanation: An MQPUT call was issued with the MQPMO_SET_IDENTITY_CONTEXT option specified in the *PutMsgOpts* parameter, but the queue had not been opened with the MQOO_SET_IDENTITY_CONTEXT option.

Completion Code: MQCC_FAILED

Programmer Response: Specify MQOO_SET_IDENTITY_CONTEXT (or another option that implies it) when the queue is opened.

MQRC_OBJECT_ALREADY_EXISTS
(2100, X'0834')

Explanation: An MQOPEN call was issued to create a dynamic queue, but a queue with the same name as the dynamic queue already exists.

- On OS/390, a rare "race condition" can also give rise to this reason code; see the description of reason code MQRC_NAME_IN_USE for more details.

Completion Code: MQCC_FAILED

Programmer Response: If supplying a dynamic queue name in full, ensure that it obeys the naming conventions for dynamic queues; if it does, either supply a different name, or delete the existing queue if it is no longer required. Alternatively, allow the queue manager to generate the name.

If the queue manager is generating the name (either in part or in full), reissue the MQOPEN call.

MQRC_OBJECT_CHANGED
(2041, X'07F9')

Explanation: Object definitions that affect this object have been changed since the *Hobj* handle used on this call was returned by the MQOPEN call. See "Chapter 34. MQOPEN - Open object" on page 375 for more information.

This reason does not occur if the object handle is specified in the *Context* field of the *PutMsgOpts* parameter on the MQPUT or MQPUT1 call.

Completion Code: MQCC_FAILED

Programmer Response: Issue an MQCLOSE call to return the handle to the system. It is then usually sufficient to reopen the object and retry the operation. However, if the object definitions are critical to the application logic, an MQINQ call can be used after reopening the object, to obtain the new values of the object attributes.

MQRC_OBJECT_DAMAGED
(2101, X'0835')

Explanation: The object accessed by the call is damaged and cannot be used. For example, this may be because the definition of the object in main storage is not consistent, or because it differs from the definition of the object on disk, or because the definition on disk cannot be read.

The object cannot be used until the problem is corrected. The object can be deleted, although it may not be possible to delete the associated user space.

- On OS/390, this reason code does not occur.

Completion Code: MQCC_FAILED

Programmer Response: It may be necessary to stop and restart the queue manager, or to restore the queue-manager data from back-up storage.

- On Compaq (DIGITAL) OpenVMS, OS/2, AS/400, Tandem NonStop Kernel, and UNIX systems, consult the FFST™ record to obtain more detail about the problem.

MQRC_OBJECT_IN_USE
(2042, X'07FA')

Explanation: An MQOPEN call was issued, but the object in question has already been opened by this or another application with options that conflict with those specified in the *Options* parameter. This arises if the request is for shared input, but the object is already open for exclusive input; it also arises if the request is for exclusive input, but the object is already open for input (of any sort).

MCAs for receiver channels, or the intra-group queuing agent (IGQ agent), may keep the destination queues open even when messages are not being transmitted; this results in the queues appearing to be"in use". Use the MQSC command DISPLAY QSTATUS to find out who is keeping the queue open.

- On OS/390, this reason can also occur for an MQOPEN or MQPUT1 call, if the object to be opened (which can be a queue, or for MQOPEN a namelist or process object) is in the process of being deleted.

Completion Code: MQCC_FAILED

Programmer Response: System design should specify whether an application is to wait and retry, or take other action.

MQRC_OBJECT_LEVEL_INCOMPATIBLE
(2360, X'0938')

Explanation: An MQOPEN or MQPUT1 call was issued, but the definition of the object to be accessed is not compatible with the queue manager to which the application has connected. The object definition was created or modified by a different version of the queue manager.

If the object to be accessed is a queue, the incompatible object definition could be the object specified by the MQOD structure on the call, or one of the object definitions used to resolve the specified object (for example, the base queue to which an alias queue resolves, or the transmission queue to which a remote queue or queue-manager alias resolves).

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: The application must be run on a queue manager that is compatible with the object definition. Refer to the *MQSeries for OS/390 Concepts and Planning Guide* and the *MQSeries for OS/390 System Setup Guide* for information about compatibility and migration between different versions of the queue manager.

MQRC_OBJECT_NAME_ERROR
(2152, X'0868')

Explanation: An MQOPEN or MQPUT1 call was issued to open a distribution list (that is, the *RecsPresent* field in MQOD is greater than zero), but the *ObjectName* field is neither blank nor the null string.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: If it is intended to open a distribution list, set the *ObjectName* field to blanks or the null string. If it is not intended to open a distribution list, set the *RecsPresent* field to zero.

MQRC_OBJECT_NOT_UNIQUE
(2343, X'0927')

Explanation: An MQOPEN or MQPUT1 call was issued to access a queue, but the call failed because the queue specified in the MQOD structure cannot be resolved unambiguously. There exists a shared queue with the specified name, and a nonshared queue with the same name.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: One of the queues must be deleted. If the queue to be deleted contains messages, use the MQSC command MOVE QLOCAL to move the messages to a different queue, and then use the command DELETE QLOCAL to delete the queue.

MQRC_OBJECT_Q_MGR_NAME_ERROR
(2153, X'0869')

Explanation: An MQOPEN or MQPUT1 call was issued to open a distribution list (that is, the *RecsPresent* field in MQOD is greater than zero), but the *ObjectQMgrName* field is neither blank nor the null string.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: If it is intended to open a distribution list, set the *ObjectQMgrName* field to blanks or the null string. If it is not intended to open a distribution list, set the *RecsPresent* field to zero.

MQRC_OBJECT_RECORDS_ERROR
(2155, X'086B')

Explanation: An MQOPEN or MQPUT1 call was issued to open a distribution list (that is, the *RecsPresent* field in MQOD is greater than zero), but the MQOR object records are not specified correctly. One of the following applies:
- *ObjectRecOffset* is zero and *ObjectRecPtr* is zero or the null pointer.
- *ObjectRecOffset* is not zero and *ObjectRecPtr* is not zero and not the null pointer.
- *ObjectRecPtr* is not a valid pointer.
- *ObjectRecPtr* or *ObjectRecOffset* points to storage that is not accessible.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that one of *ObjectRecOffset* and *ObjectRecPtr* is zero and the other nonzero. Ensure that the field used points to accessible storage.

MQRC_OBJECT_TYPE_ERROR
(2043, X'07FB')

Explanation: On the MQOPEN or MQPUT1 call, the *ObjectType* field in the object descriptor MQOD specifies a value that is not valid. For the MQPUT1 call, the object type must be MQOT_Q.

Completion Code: MQCC_FAILED

Programmer Response: Specify a valid object type.

MQRC_OD_ERROR
(2044, X'07FC')

Explanation: On the MQOPEN or MQPUT1 call, the object descriptor MQOD is not valid, for one of the following reasons:
- The *StrucId* field is not MQOD_STRUC_ID.
- The *Version* field specifies a value that is not valid or not supported.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The queue manager cannot copy the changed structure to application storage, even though the call is successful. This can occur, for example, if the pointer points to read-only storage.

Completion Code: MQCC_FAILED

Programmer Response: Correct the definition of the object descriptor. Ensure that required input fields are set correctly.

MQRC_OFFSET_ERROR
(2251, X'08CB')

Explanation: An MQPUT or MQPUT1 call was issued, but the value of the *Offset* field in the MQMD or MQMDE structure is less than zero or greater than 999 999 999.

This error can also occur on the MQPUT call if the *Offset* field would have become greater than 999 999 999 as a result of the call.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Specify a value in the range 0 through 999 999 999. Do not attempt to create a message segment that would extend beyond an offset of 999 999 999.

MQRC_OPEN_FAILED
(2137, X'0859')

Explanation: A queue or other MQ object could not be opened successfully, for one of the following reasons:
- An MQCONN or MQCONNX call was issued, but the queue manager was unable to open an object that is used internally by the queue manager. As a result, processing cannot continue. The error log will contain the name of the object that could not be opened.
- An MQPUT call was issued to put a message to a distribution list, but the message could not be sent to the destination to which this reason code applies because that destination was not opened successfully by the MQOPEN call. This reason occurs only in the *Reason* field of the MQRR response record.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Do one of the following:
- If the error occurred on the MQCONN or MQCONNX call, ensure that the required objects exist by running the following command and then retrying the application:

  ```
  STRMQM -c qmgr
  ```

  where `qmgr` should be replaced by the name of the queue manager.
- If the error occurred on the MQPUT call, examine the MQRR response records specified on the MQOPEN call to determine the reason that the queue failed to open. Ensure that sufficient response records are provided by the application on the call to enable the error(s) to be determined.

MQRC_OPTION_ENVIRONMENT_ERROR (2274, X'08E2')

Explanation: An MQGET call with the MQGMO_MARK_SKIP_BACKOUT option specified was issued from a DB2 Stored Procedure. The call failed because the MQGMO_MARK_SKIP_BACKOUT option cannot be used from a DB2 Stored Procedure.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Remove the MQGMO_MARK_SKIP_BACKOUT option from the MQGET call.

MQRC_OPTION_NOT_VALID_FOR_TYPE (2045, X'07FD')

Explanation: On an MQOPEN or MQCLOSE call, an option is specified that is not valid for the type of object or queue being opened or closed.

For the MQOPEN call, this includes the following cases:

- An option that is inappropriate for the object type (for example, MQOO_OUTPUT for an MQOT_PROCESS object).
- An option that is unsupported for the queue type (for example, MQOO_INQUIRE for a remote queue that has no local definition).
- One or more of the following options:
  MQOO_INPUT_AS_Q_DEF
  MQOO_INPUT_SHARED
  MQOO_INPUT_EXCLUSIVE
  MQOO_BROWSE
  MQOO_INQUIRE
  MQOO_SET

  when either:
  - the queue name is resolved through a cell directory, or
  - *ObjectQMgrName* in the object descriptor specifies the name of a local definition of a remote queue (in order to specify a queue-manager alias), and the queue named in the *RemoteQMgrName* attribute of the definition is the name of the local queue manager.

For the MQCLOSE call, this includes the following case:

- The MQCO_DELETE or MQCO_DELETE_PURGE option when the queue is not a dynamic queue.

This reason code can also occur on the MQOPEN call when the object being opened is of type MQOT_NAMELIST, MQOT_PROCESS, or MQOT_Q_MGR, but the *ObjectQMgrName* field in MQOD is neither blank nor the name of the local queue manager.

Completion Code: MQCC_FAILED

Programmer Response: Specify the correct option; see Table 74 on page 381 for open options, and Table 69 on page 321 for close options. For the MQOPEN call, ensure that the *ObjectQMgrName* field is set correctly. For the MQCLOSE call, either correct the option or change the definition type of the model queue that is used to create the new queue.

MQRC_OPTIONS_ERROR (2046, X'07FE')

Explanation: The *Options* parameter or field contains options that are not valid, or a combination of options that is not valid.

- For the MQOPEN, MQCLOSE, MQXCNVC, mqBagToBuffer, mqBufferToBag, mqCreateBag, and mqExecute calls, *Options* is a separate parameter on the call.

  This reason also occurs if the parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- For the MQBEGIN, MQCONNX, MQGET, MQPUT, and MQPUT1 calls, *Options* is a field in the relevant options structure (MQBO, MQCNO, MQGMO, or MQPMO).

Completion Code: MQCC_FAILED

Programmer Response: Specify valid options. Check the description of the *Options* parameter or field to determine which options and combinations of options are valid. If multiple options are being set by adding the individual options together, ensure that the same option is not added twice.

MQRC_ORIGINAL_LENGTH_ERROR (2252, X'08CC')

Explanation: An MQPUT or MQPUT1 call was issued to put a report message that is a segment, but the *OriginalLength* field in the MQMD or MQMDE structure is either:

- Less than one (for a segment that is not the last segment), or
- Less than zero (for a segment that is the last segment)

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Specify a value that is greater than zero. Zero is valid only for the last segment.

MQRC_OUT_SELECTOR_ERROR
(2310, X'0906')

Explanation: The *OutSelector* parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code: MQCC_FAILED

Programmer Response: Correct the parameter.

MQRC_OUTCOME_MIXED
(2123, X'084B')

Explanation: The queue manager is acting as the unit-of-work coordinator for a unit of work that involves other resource managers, but one of the following occurred:

- An MQCMIT or MQDISC call was issued to commit the unit of work, but one or more of the participating resource managers backed-out the unit of work instead of committing it. As a result, the outcome of the unit of work is mixed.
- An MQBACK call was issued to back out a unit of work, but one or more of the participating resource managers had already committed the unit of work.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, Sun Solaris, Windows NT.

Completion Code: MQCC_FAILED

Programmer Response: Examine the queue-manager error logs for messages relating to the mixed outcome; these messages identify the resource managers that are affected. Use procedures local to the affected resource managers to resynchronize the resources.

This reason code does not prevent the application initiating further units of work.

MQRC_OUTCOME_PENDING
(2124, X'084C')

Explanation: The queue manager is acting as the unit-of-work coordinator for a unit of work that involves other resource managers, and an MQCMIT or MQDISC call was issued to commit the unit of work, but one or more of the participating resource managers has not confirmed that the unit of work was committed successfully.

The completion of the commit operation will happen at some point in the future, but there remains the possibility that the outcome will be mixed.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, Sun Solaris, Windows NT.

Completion Code: MQCC_WARNING

Programmer Response: Use the normal error-reporting mechanisms to determine whether the outcome was mixed. If it was, take appropriate action to resynchronize the resources.

This reason code does not prevent the application initiating further units of work.

MQRC_PAGESET_ERROR
(2193, X'0891')

Explanation: An error was encountered with the page set while attempting to access it for a locally defined queue. This could be because the queue is on a page set that does not exist. A console message is issued that tells you the number of the page set in error. For example if the error occurred in the TEST job, and your user identifier is ABCDEFG, the message is:

```
CSQI041I CSQIALLC JOB TEST USER ABCDEFG HAD ERROR ACCESSING PAGE
```

If this reason code occurs while attempting to delete a dynamic queue with MQCLOSE, the dynamic queue has not been deleted.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Check that the storage class for the queue maps to a valid page set using the DISPLAY Q(xx) STGCLASS, DISPLAY STGCLASS(xx), and DISPLAY USAGE PSID commands. If you are unable to resolve the problem, notify the system programmer who should:

- Collect the following diagnostic information:
  - A description of the actions that led to the error
  - A listing of the application program being run at the time of the error
  - Details of the page sets defined for use by MQSeries
- Attempt to re-create the problem, and take a system dump immediately after the error occurs
- Contact your IBM Support Center

MQRC_PAGESET_FULL
(2192, X'0890')

Explanation: An MQI call was issued to operate on a queue, but the call failed because the external storage medium is full. One of the following applies:

- A page-set data set is full (nonshared queues only).
- A coupling-facility structure is full (shared queues only).

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Check which queues contain messages and look for applications that might be filling the queues unintentionally. Be aware that the queue that has caused the page set or coupling-facility structure to become full is not necessarily the queue

| referenced by the MQI call that returned
| MQRC_PAGESET_FULL.

| Check that all of the usual server applications are
| operating correctly and processing the messages on the
| queues.

| If the applications and servers are operating correctly,
| increase the number of server applications to cope with
| the message load, or request the system programmer to
| increase the size of the page-set data sets.

MQRC_PARAMETER_MISSING
(2321, X'0911')

Explanation: An administration message requires a parameter that is not present in the administration bag. This reason code occurs only for bags created with the MQCBO_ADMIN_BAG or MQCBO_REORDER_AS_REQUIRED options.

Completion Code: MQCC_FAILED

Programmer Response: Review the description of the administration command being issued, and ensure that all required parameters are present in the bag.

MQRC_PARTICIPANT_NOT_AVAILABLE
(2122, X'084A')

Explanation: An MQBEGIN call was issued to start a unit of work coordinated by the queue manager, but one or more of the participating resource managers that had been registered with the queue manager is not available. As a result, changes to those resources cannot be coordinated by the queue manager in the unit of work.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT.

Completion Code: MQCC_WARNING

Programmer Response: If the application does not require non-MQ resources to participate in the unit of work, this reason code can be ignored. Otherwise consult your system support programmer to determine why the required resource managers are not available. The resource manager may have been halted temporarily, or there may be an error in the queue manager's configuration file.

MQRC_PCF_ERROR
(2149, X'0865')

Explanation: An MQPUT or MQPUT1 call was issued to put a message containing PCF data, but the length of the message does not equal the sum of the lengths of the PCF structures present in the message. This can occur for messages with the following format names:

MQFMT_ADMIN
MQFMT_EVENT
MQFMT_PCF

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the length of the message specified on the MQPUT or MQPUT1 call equals the sum of the lengths of the PCF structures contained within the message data.

MQRC_PERSISTENCE_ERROR
(2047, X'07FF')

Explanation: On an MQPUT or MQPUT1 call, the value specified for the *Persistence* field in the message descriptor MQMD is not valid.

Completion Code: MQCC_FAILED

Programmer Response: Specify one of the following values:

MQPER_PERSISTENT
MQPER_NOT_PERSISTENT
MQPER_PERSISTENCE_AS_Q_DEF

MQRC_PERSISTENT_NOT_ALLOWED
(2048, X'0800')

Explanation: On an MQPUT or MQPUT1 call, the value specified for the *Persistence* field in MQMD (or obtained from the *DefPersistence* queue attribute) specifies MQPER_PERSISTENT, but the queue on which the message is being placed does not support persistent messages. Persistent messages cannot be placed on:

- Temporary dynamic queues
- Shared queues

This reason code can also occur in the *Feedback* field in the message descriptor of a report message; in this case it indicates that the error was encountered by a message channel agent when it attempted to put the message on a remote queue.

Completion Code: MQCC_FAILED

Programmer Response: Specify MQPER_NOT_PERSISTENT if the message is to be placed on a temporary dynamic queue or shared queue. If persistence is required, use a permanent dynamic queue or predefined queue in place of a temporary dynamic queue.

Be aware that server applications are recommended to send reply messages (message type MQMT_REPLY) with the same persistence as the original request message (message type MQMT_REQUEST). If the request message is persistent, the reply queue specified in the *ReplyToQ* field in the message descriptor MQMD cannot be a temporary dynamic queue; a permanent dynamic or predefined queue must be used as the reply queue in this situation.

MQRC_PMO_ERROR
(2173, X'087D')

Explanation: On an MQPUT or MQPUT1 call, the MQPMO structure is not valid, for one of the following reasons:
- The *StrucId* field is not MQPMO_STRUC_ID.
- The *Version* field specifies a value that is not valid or not supported.
- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The queue manager cannot copy the changed structure to application storage, even though the call is successful. This can occur, for example, if the pointer points to read-only storage.

Completion Code: MQCC_FAILED

Programmer Response: Correct the definition of the MQPMO structure. Ensure that required input fields are correctly set.

MQRC_PMO_RECORD_FLAGS_ERROR
(2158, X'086E')

Explanation: An MQPUT or MQPUT1 call was issued to put a message, but the *PutMsgRecFields* field in the MQPMO structure is not valid, for one of the following reasons:
- The field contains flags that are not valid.
- The message is being put to a distribution list, and put message records have been provided (that is, *RecsPresent* is greater than zero, and one of *PutMsgRecOffset* or *PutMsgRecPtr* is nonzero), but *PutMsgRecFields* has the value MQPMRF_NONE.
- MQPMRF_ACCOUNTING_TOKEN is specified without either MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that *PutMsgRecFields* is set with the appropriate MQPMRF_* flags to indicate which fields are present in the put message records. If MQPMRF_ACCOUNTING_TOKEN is specified, ensure that either MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT is also specified. Alternatively, set both *PutMsgRecOffset* and *PutMsgRecPtr* to zero.

MQRC_PRIORITY_ERROR
(2050, X'0802')

Explanation: An MQPUT or MQPUT1 call was issued, but the value of the *Priority* field in the message descriptor MQMD is not valid. The maximum priority supported by the queue manager is given by the *MaxPriority* queue-manager attribute.

Completion Code: MQCC_FAILED

Programmer Response: Specify a value in the range zero through *MaxPriority*, or the special value MQPRI_PRIORITY_AS_Q_DEF.

MQRC_PRIORITY_EXCEEDS_MAXIMUM
(2049, X'0801')

Explanation: An MQPUT or MQPUT1 call was issued, but the value of the *Priority* field in the message descriptor MQMD exceeds the maximum priority supported by the local queue manager (see the *MaxPriority* queue-manager attribute described in "Chapter 42. Attributes for the queue manager" on page 469). The message is accepted by the queue manager, but is placed on the queue at the queue manager's maximum priority. The *Priority* field in the message descriptor retains the value specified by the application that put the message.

Completion Code: MQCC_WARNING

Programmer Response: None required, unless this reason code was not expected by the application that put the message.

MQRC_PUT_INHIBITED
(2051, X'0803')

Explanation: MQPUT and MQPUT1 calls are currently inhibited for the queue, or for the queue to which this queue resolves. See the *InhibitPut* queue attribute described in "Chapter 39. Attributes for queues" on page 427.

This reason code can also occur in the *Feedback* field in the message descriptor of a report message; in this case it indicates that the error was encountered by a message channel agent when it attempted to put the message on a remote queue.

Completion Code: MQCC_FAILED

Programmer Response: If the system design allows put requests to be inhibited for short periods, retry the operation later.

MQRC_PUT_MSG_RECORDS_ERROR
(2159, X'086F')

Explanation: An MQPUT or MQPUT1 call was issued to put a message to a distribution list, but the MQPMR put message records are not specified correctly. One of the following applies:
- *PutMsgRecOffset* is not zero and *PutMsgRecPtr* is not zero and not the null pointer.
- *PutMsgRecPtr* is not a valid pointer.
- *PutMsgRecPtr* or *PutMsgRecOffset* points to storage that is not accessible.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that at least one of *PutMsgRecOffset* and *PutMsgRecPtr* is zero. Ensure that the field used points to accessible storage.

MQRC_Q_ALREADY_EXISTS
(2290, X'08F2')

Explanation: This reason should be returned by the MQZ_INSERT_NAME installable service component when the queue specified by the *QName* parameter is already defined to the name service.

- On OS/390, this reason code does not occur.

Completion Code: MQCC_FAILED

Programmer Response: None. See the *MQSeries Programmable System Management* book for information about installable service.

MQRC_Q_DELETED
(2052, X'0804')

Explanation: An *Hobj* queue handle specified on a call refers to a dynamic queue that has been deleted since the queue was opened. (See "Chapter 27. MQCLOSE - Close object" on page 319 for information about the deletion of dynamic queues.)

- On OS/390, this can also occur with the MQOPEN and MQPUT1 calls if a dynamic queue is being opened, but the queue is in a logically-deleted state. See MQCLOSE for more information about this.

Completion Code: MQCC_FAILED

Programmer Response: Issue an MQCLOSE call to return the handle and associated resources to the system (the MQCLOSE call will succeed in this case). Check the design of the application that caused the error.

MQRC_Q_DEPTH_HIGH
(2224, X'08B0')

Explanation: An MQPUT or MQPUT1 call has caused the queue depth to be incremented to or above the limit specified in the *QDepthHighLimit* attribute.

Completion Code: MQCC_WARNING

Programmer Response: None. This reason code is only used to identify the corresponding event message.

MQRC_Q_DEPTH_LOW
(2225, X'08B1')

Explanation: An MQGET call has caused the queue depth to be decremented to or below the limit specified in the *QDepthLowLimit* attribute.

Completion Code: MQCC_WARNING

Programmer Response: None. This reason code is only used to identify the corresponding event message.

MQRC_Q_FULL
(2053, X'0805')

Explanation: On an MQPUT or MQPUT1 call, the call failed because the queue is full, that is, it already contains the maximum number of messages possible (see the *MaxQDepth* queue attribute described in "Chapter 39. Attributes for queues" on page 427).

This reason code can also occur in the *Feedback* field in the message descriptor of a report message; in this case it indicates that the error was encountered by a message channel agent when it attempted to put the message on a remote queue.

Completion Code: MQCC_FAILED

Programmer Response: Retry the operation later. Consider increasing the maximum depth for this queue, or arranging for more instances of the application to service the queue.

MQRC_Q_MGR_ACTIVE
(2222, X'08AE')

Explanation: This condition is detected when a queue manager becomes active.

- On OS/390, this event is not generated for the first start of a queue manager, only on subsequent restarts.

Completion Code: MQCC_WARNING

Programmer Response: None. This reason code is only used to identify the corresponding event message.

MQRC_Q_MGR_NAME_ERROR
(2058, X'080A')

Explanation: On an MQCONN or MQCONNX call, the value specified for the *QMgrName* parameter is not valid or not known. This reason also occurs if the parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

- On OS/390 for CICS applications, this reason can occur on *any* call if the original connect specified an incorrect or unrecognized name.

This reason code can also occur if an MQ client application attempts to connect to a queue manager within an MQ-client queue-manager group (see the *QMgrName* parameter of MQCONN), and either:

- Queue-manager groups are not supported.
- There is no queue-manager group with the specified name.

Completion Code: MQCC_FAILED

Programmer Response: Use an all-blank name if possible, or verify that the name used is valid.

MQRC_Q_MGR_NOT_ACTIVE
(2223, X'08AF')

Explanation: This condition is detected when a queue manager is requested to stop or quiesce.

Completion Code: MQCC_WARNING

Programmer Response: None. This reason code is only used to identify the corresponding event message.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'080B')

Explanation: On an MQCONN or MQCONNX call, the queue manager identified by the *QMgrName* parameter is not available for connection.

- On OS/390:
  - For batch applications, this reason can be returned to applications running in non-MQSeries LPARs.
  - For CICS applications, this reason can occur on *any* call if the original connect specified a queue manager whose name was recognized, but which is not available.
- On AS/400, this reason can also be returned by the MQOPEN and MQPUT1 calls, when MQHC_DEF_HCONN is specified for the *Hconn* parameter by an application running in compatibility mode.

This reason code can also occur if an MQ client application attempts to connect to a queue manager within an MQ-client queue-manager group when none of the queue managers in the group is available for connection (see the *QMgrName* parameter of the MQCONN call).

This reason code can also occur if the call is issued by an MQ client application and there is an error with the client-connection or the corresponding server-connection channel definitions.

- On OS/390, this reason code can also occur if the optional OS/390 client attachment feature has not been installed.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the queue manager has been started. If the connection is from a client application, check the channel definitions.

MQRC_Q_MGR_QUIESCING
(2161, X'0871')

Explanation: An MQI call was issued, but the call failed because the queue manager is quiescing (preparing to shut down).

When the queue manager is quiescing, the MQOPEN, MQPUT, MQPUT1, and MQGET calls can still complete successfully, but the application can request that they fail by specifying the appropriate option on the call:

- MQOO_FAIL_IF_QUIESCING on MQOPEN
- MQPMO_FAIL_IF_QUIESCING on MQPUT or MQPUT1
- MQGMO_FAIL_IF_QUIESCING on MQGET

Specifying these options enables the application to become aware that the queue manager is preparing to shut down.

- On OS/390:
  - For batch applications, this reason can be returned to applications running in non-MQSeries LPARs.
  - For CICS applications, this reason can be returned when no connection was established.
- On AS/400 for applications running in compatibility mode, this reason can be returned when no connection was established.

Completion Code: MQCC_FAILED

Programmer Response: The application should tidy up and end. If the application specified the MQOO_FAIL_IF_QUIESCING, MQPMO_FAIL_IF_QUIESCING, or MQGMO_FAIL_IF_QUIESCING option on the failing call, the relevant option can be removed and the call reissued. By omitting these options, the application can continue working in order to complete and commit the current unit of work, but the application should not start a new unit of work.

MQRC_Q_MGR_STOPPING
(2162, X'0872')

Explanation: An MQI call was issued, but the call failed because the queue manager is shutting down. If the call was an MQGET call with the MQGMO_WAIT option, the wait has been canceled. No more MQI calls can be issued.

For MQ client applications, it is possible that the call did complete successfully, even though this reason code is returned with a *CompCode* of MQCC_FAILED.

- On OS/390, the MQRC_CONNECTION_BROKEN reason may be returned instead if, as a result of system scheduling factors, the queue manager shuts down before the call completes.

Completion Code: MQCC_FAILED

Programmer Response: The application should tidy up and end. If the application is in the middle of a unit of work coordinated by an external unit-of-work coordinator, the application should issue the appropriate call to back out the unit of work. Any unit of work that is coordinated by the queue manager is backed out automatically.

MQRC_Q_NOT_EMPTY
(2055, X'0807')

Explanation: An MQCLOSE call was issued for a permanent dynamic queue, but the call failed because

the queue is not empty or still in use. One of the following applies:
- The MQCO_DELETE option was specified, but there are messages on the queue.
- The MQCO_DELETE or MQCO_DELETE_PURGE option was specified, but there are uncommitted get or put calls outstanding against the queue.

See the usage notes pertaining to dynamic queues for the MQCLOSE call for more information.

This reason code is also returned from a Programmable Command Format (PCF) command to clear or delete a queue, if the queue contains uncommitted messages (or committed messages in the case of delete queue without the purge option).

Completion Code: MQCC_FAILED

Programmer Response: Check why there might be messages on the queue. Be aware that the *CurrentQDepth* queue attribute might be zero even though there are one or more messages on the queue; this can happen if the messages have been retrieved as part of a unit of work that has not yet been committed. If the messages can be discarded, try using the MQCLOSE call with the MQCO_DELETE_PURGE option. Consider retrying the call later.

MQRC_Q_SERVICE_INTERVAL_HIGH
(2226, X'08B2')

Explanation: No successful gets or puts have been detected within an interval that is greater than the limit specified in the *QServiceInterval* attribute.

Completion Code: MQCC_WARNING

Programmer Response: None. This reason code is only used to identify the corresponding event message.

MQRC_Q_SERVICE_INTERVAL_OK
(2227, X'08B3')

Explanation: A successful get has been detected within an interval that is less than or equal to the limit specified in the *QServiceInterval* attribute.

Completion Code: MQCC_WARNING

Programmer Response: None. This reason code is only used to identify the corresponding event message.

MQRC_Q_SPACE_NOT_AVAILABLE
(2056, X'0808')

Explanation: An MQPUT or MQPUT1 call was issued, but there is no space available for the queue on disk or other storage device.

This reason code can also occur in the *Feedback* field in the message descriptor of a report message; in this case it indicates that the error was encountered by a message channel agent when it attempted to put the message on a remote queue.
- On OS/390, this reason code does not occur.

Completion Code: MQCC_FAILED

Programmer Response: Check whether an application is putting messages in an infinite loop. If not, make more disk space available for the queue.

MQRC_Q_TYPE_ERROR
(2057, X'0809')

Explanation: One of the following occurred:
- On an MQOPEN call, the *ObjectQMgrName* field in the object descriptor MQOD or object record MQOR specifies the name of a local definition of a remote queue (in order to specify a queue-manager alias), and in that local definition the *RemoteQMgrName* attribute is the name of the local queue manager. However, the *ObjectName* field in MQOD or MQOR specifies the name of a model queue on the local queue manager; this is not allowed. See the *MQSeries Application Programming Guide* for more information.
- On an MQPUT1 call, the object descriptor MQOD or object record MQOR specifies the name of a model queue.
- On a previous MQPUT or MQPUT1 call, the *ReplyToQ* field in the message descriptor specified the name of a model queue, but a model queue cannot be specified as the destination for reply or report messages. Only the name of a predefined queue, or the name of the *dynamic* queue created from the model queue, can be specified as the destination. In this situation the reason code MQRC_Q_TYPE_ERROR is returned in the *Reason* field of the MQDLH structure when the reply message or report message is placed on the dead-letter queue.

Completion Code: MQCC_FAILED

Programmer Response: Specify a valid queue.

MQRC_RECS_PRESENT_ERROR
(2154, X'086A')

Explanation: An MQOPEN or MQPUT1 call was issued, but the call failed for one of the following reasons:
- *RecsPresent* in MQOD is less than zero.
- *ObjectType* in MQOD is not MQOT_Q, and *RecsPresent* is not zero. *RecsPresent* must be zero if the object being opened is not a queue.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: If it is intended to open a distribution list, set the *ObjectType* field to MQOT_Q and *RecsPresent* to the number of destinations in the list. If it is not intended to open a distribution list, set

the *RecsPresent* field to zero.

MQRC_REMOTE_Q_NAME_ERROR (2184, X'0888')

Explanation: On an MQOPEN or MQPUT1 call, one of the following occurred:

- A local definition of a remote queue (or an alias to one) was specified, but the *RemoteQName* attribute in the remote queue definition is entirely blank. Note that this error occurs even if the *XmitQName* in the definition is not blank.
- The *ObjectQMgrName* field in the object descriptor was not blank and not the name of the local queue manager, but the *ObjectName* field is blank.

Completion Code: MQCC_FAILED

Programmer Response: Alter the local definition of the remote queue and supply a valid remote queue name, or supply a nonblank *ObjectName* in the object descriptor, as appropriate.

MQRC_REPORT_OPTIONS_ERROR (2061, X'080D')

Explanation: An MQPUT or MQPUT1 call was issued, but the *Report* field in the message descriptor MQMD contains one or more options that are not recognized by the local queue manager. The options that cause this reason code to be returned depend on the destination of the message; see "Appendix E. Report options and message flags" on page 593 for more details.

This reason code can also occur in the *Feedback* field in the MQMD of a report message, or in the *Reason* field in the MQDLH structure of a message on the dead-letter queue; in both cases it indicates that the destination queue manager does not support one or more of the report options specified by the sender of the message.

Completion Code: MQCC_FAILED

Programmer Response: Do the following:

- Ensure that the *Report* field in the message descriptor is initialized with a value when the message descriptor is declared, or is assigned a value prior to the MQPUT or MQPUT1 call. Specify MQRO_NONE if no report options are required.
- Ensure that the report options specified are ones that are documented in this book; see the *Report* field described in "Chapter 9. MQMD - Message descriptor" on page 125 for valid report options. Remove any report options that are not documented in this book.
- If multiple report options are being set by adding the individual report options together, ensure that the same report option is not added twice.
- Check that conflicting report options are not specified. For example, do not add both MQRO_EXCEPTION and MQRO_EXCEPTION_WITH_DATA to the *Report* field; only one of these can be specified.

MQRC_RESOURCE_PROBLEM (2102, X'0836')

Explanation: There are insufficient system resources to complete the call successfully.

Completion Code: MQCC_FAILED

Programmer Response: Run the application when the machine is less heavily loaded.

- On OS/390, check the operator console for messages that may provide additional information.
- On Compaq (DIGITAL) OpenVMS, OS/2, AS/400, Tandem NonStop Kernel, and UNIX systems, consult the FFST™ record to obtain more detail about the problem.

MQRC_RESPONSE_RECORDS_ERROR (2156, X'086C')

Explanation: An MQOPEN or MQPUT1 call was issued to open a distribution list (that is, the *RecsPresent* field in MQOD is greater than zero), but the MQRR response records are not specified correctly. One of the following applies:

- *ResponseRecOffset* is not zero and *ResponseRecPtr* is not zero and not the null pointer.
- *ResponseRecPtr* is not a valid pointer.
- *ResponseRecPtr* or *ResponseRecOffset* points to storage that is not accessible.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that at least one of *ResponseRecOffset* and *ResponseRecPtr* is zero. Ensure that the field used points to accessible storage.

MQRC_RFH_COMMAND_ERROR (2336, X'0920')

Explanation: The message contains an MQRFH structure, but the command name contained in the *NameValueString* field is not valid.

Completion Code: MQCC_FAILED

Programmer Response: Modify the application that generated the message to ensure that it places in the *NameValueString* field a command name that is valid.

MQRC_RFH_DUPLICATE_PARM (2338, X'0922')

Explanation: The message contains an MQRFH structure, but a parameter occurs more than once in the *NameValueString* field when only one occurrence is

valid for the specified command.

Completion Code: MQCC_FAILED

Programmer Response: Modify the application that generated the message to ensure that it places in the *NameValueString* field only one occurrence of the parameter.

MQRC_RFH_ERROR
(2334, X'091E')

Explanation: The message contains an MQRFH structure, but the structure is not valid.

Completion Code: MQCC_FAILED

Programmer Response: Modify the application that generated the message to ensure that it places a valid MQRFH structure in the message data.

MQRC_RFH_PARM_ERROR
(2337, X'0921')

Explanation: The message contains an MQRFH structure, but a parameter name contained in the *NameValueString* field is not valid for the command specified.

Completion Code: MQCC_FAILED

Programmer Response: Modify the application that generated the message to ensure that it places in the *NameValueString* field only parameters that are valid for the specified command.

MQRC_RFH_PARM_MISSING
(2339, X'0923')

Explanation: The message contains an MQRFH structure, but the command specified in the *NameValueString* field requires a parameter that is not present.

Completion Code: MQCC_FAILED

Programmer Response: Modify the application that generated the message to ensure that it places in the *NameValueString* field all parameters that are required for the specified command.

MQRC_RFH_STRING_ERROR
(2335, X'091F')

Explanation: The contents of the *NameValueString* field in the MQRFH structure are not valid. *NameValueString* must adhere to the following rules:

- The string must consist of zero or more name/value pairs separated from each other by one or more blanks; the blanks are not significant.
- If a name or value contains blanks that are significant, the name or value must be enclosed in double-quote characters.
- If a name or value itself contains one or more double-quote characters, the name or value must be enclosed in double-quote characters, and each embedded double-quote character must be doubled.
- A name or value can contain any characters other than the null, which acts as a delimiter. The null and characters following it, up to the defined length of *NameValueString*, are ignored.

The following is a valid *NameValueString*:

```
Famous_Words "The program displayed ""Hello World"""
```

Completion Code: MQCC_FAILED

Programmer Response: Modify the application that generated the message to ensure that it places in the *NameValueString* field data that adheres to the rules listed above. Check that the *StrucLength* field is set to the correct value.

MQRC_RMH_ERROR
(2220, X'08AC')

Explanation: On an MQPUT or MQPUT1 call, the reference message header structure MQRMH in the message data is not valid.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Correct the definition of the MQRMH structure. Ensure that the fields are set correctly.

MQRC_SECOND_MARK_NOT_ALLOWED
(2062, X'080E')

Explanation: An MQGET call was issued specifying the MQGMO_MARK_SKIP_BACKOUT option in the *Options* field of MQGMO, but a message has already been marked within the current unit of work. Only one marked message is allowed within each unit of work.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Modify the application so that no more than one message is marked within each unit of work.

MQRC_SECURITY_ERROR
(2063, X'080F')

Explanation: An MQCONN, MQCONNX, MQOPEN, MQPUT1, or MQCLOSE call was issued, but it failed because a security error occurred.

- On OS/390, the security error was returned by the External Security Manager.

Completion Code: MQCC_FAILED

Programmer Response: Note the error from the security manager, and contact your system programmer or security administrator.

- On AS/400, the FFST log will contain the error information.

MQRC_SEGMENT_LENGTH_ZERO
(2253, X'08CD')

Explanation: An MQPUT or MQPUT1 call was issued to put the first or an intermediate segment of a logical message, but the length of the application message data in the segment (excluding any MQ headers that may be present) is zero. The length must be at least one for the first or intermediate segment.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Check the application logic to ensure that segments are put with a length of one or greater. Only the last segment of a logical message is permitted to have a length of zero.

MQRC_SELECTOR_COUNT_ERROR
(2065, X'0811')

Explanation: On an MQINQ or MQSET call, the *SelectorCount* parameter specifies a value that is not valid. This reason also occurs if the parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code: MQCC_FAILED

Programmer Response: Specify a value in the range 0 through 256.

MQRC_SELECTOR_ERROR
(2067, X'0813')

Explanation: An MQINQ or MQSET call was issued, but the *Selectors* array contains a selector that is not valid for one of the following reasons:

- The selector is not supported or out of range.
- The selector is not applicable to the type of object whose attributes are being inquired or set.
- The selector is for an attribute that cannot be set.

This reason also occurs if the parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the value specified for the selector is valid for the object type represented by *Hobj*. For the MQSET call, also ensure that the selector represents an integer attribute that can be set.

MQRC_SELECTOR_LIMIT_EXCEEDED
(2066, X'0812')

Explanation: On an MQINQ or MQSET call, the *SelectorCount* parameter specifies a value that is larger than the maximum supported (256).

Completion Code: MQCC_FAILED

Programmer Response: Reduce the number of selectors specified on the call; the valid range is 0 through 256.

MQRC_SELECTOR_NOT_FOR_TYPE
(2068, X'0814')

Explanation: On the MQINQ call, one or more selectors in the *Selectors* array is not applicable to the type of the queue whose attributes are being inquired.

This reason also occurs when the queue is a cluster queue that resolved to a remote instance of the queue. In this case only a subset of the attributes that are valid for local queues can be inquired. See the usage notes in "Chapter 33. MQINQ - Inquire about object attributes" on page 363 for further details.

The call completes with MQCC_WARNING, with the attribute values for the inapplicable selectors set as follows:

- For integer attributes, the corresponding elements of *IntAttrs* are set to MQIAV_NOT_APPLICABLE.
- For character attributes, the appropriate parts of the *CharAttrs* string are set to a character string consisting entirely of asterisks (*).

Completion Code: MQCC_WARNING

Programmer Response: Verify that the selector specified is the one that was intended.

If the queue is a cluster queue, specifying one of the MQOO_BROWSE, MQOO_INPUT_*, or MQOO_SET options in addition to MQOO_INQUIRE forces the queue to resolve to the local instance of the queue. However, if there is no local instance of the queue the MQOPEN call fails.

MQRC_SELECTOR_NOT_PRESENT
(2309, X'0905')

Explanation: The *Selector* parameter specifies a selector that does not exist in the bag.

Completion Code: MQCC_FAILED

Programmer Response: Specify a selector that does exist in the bag.

MQRC_SELECTOR_NOT_SUPPORTED
(2318, X'090E')

Explanation: The *Selector* parameter specifies a value that is a system selector (a value that is negative), but the system selector is not one that is supported by the call.

Completion Code: MQCC_FAILED

Programmer Response: Specify a selector value that is supported.

MQRC_SELECTOR_NOT_UNIQUE
(2305, X'0901')

Explanation: The *ItemIndex* parameter has the value MQIND_NONE, but the bag contains more than one data item with the selector value specified by the *Selector* parameter. MQIND_NONE requires that the bag contain only one occurrence of the specified selector.

This reason code also occurs on the mqExecute call when the administration bag contains two or more occurrences of a selector for a required parameter that permits only one occurrence.

Completion Code: MQCC_FAILED

Programmer Response: Check the logic of the application that created the bag. If correct, specify for *ItemIndex* a value that is zero or greater, and add application logic to process all of the occurrences of the selector in the bag.

Review the description of the administration command being issued, and ensure that all required parameters are defined correctly in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
(2304, X'0900')

Explanation: The *Selector* parameter has a value that is outside the valid range for the call. If the bag was created with the MQCBO_CHECK_SELECTORS option:

- For the mqAddInteger call, the value must be within the range MQIA_FIRST through MQIA_LAST.
- For the mqAddString call, the value must be within the range MQCA_FIRST through MQCA_LAST.

If the bag was not created with the MQCBO_CHECK_SELECTORS option:

- The value must be zero or greater.

Completion Code: MQCC_FAILED

Programmer Response: Specify a valid value.

MQRC_SELECTOR_TYPE_ERROR
(2299, X'08FB')

Explanation: The *Selector* parameter has the wrong data type; it must be of type Long.

Completion Code: MQCC_FAILED

Programmer Response: Declare the *Selector* parameter as Long.

MQRC_SELECTOR_WRONG_TYPE
(2312, X'0908')

Explanation: A data item with the specified selector exists in the bag, but has a data type that conflicts with the data type implied by the call being used. For example, the data item might have an integer data type, but the call being used might be mqSetString, which implies a character data type.

This reason code also occurs on the mqBagToBuffer, mqExecute, and mqPutBag calls when mqAddString or mqSetString was used to add the MQIACF_INQUIRY data item to the bag.

Completion Code: MQCC_FAILED

Programmer Response: For the mqSetInteger and mqSetString calls, specify MQIND_ALL for the *ItemIndex* parameter to delete from the bag all existing occurrences of the specified selector before creating the new occurrence with the required data type.

For the mqInquireBag, mqInquireInteger, and mqInquireString calls, use the mqInquireItemInfo call to determine the data type of the item with the specified selector, and then use the appropriate call to determine the value of the data item.

For the mqBagToBuffer, mqExecute, and mqPutBag calls, ensure that the MQIACF_INQUIRY data item is added to the bag using the mqAddInteger or mqSetInteger calls.

MQRC_SERVICE_ERROR
(2289, X'08F1')

Explanation: This reason should be returned by an installable service component when the component encounters an unexpected error.

- On OS/390, this reason code does not occur.

Completion Code: MQCC_FAILED

Programmer Response: Correct the error and retry the operation.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'08ED')

Explanation: This reason should be returned by an installable service component when the requested action cannot be performed because the required underlying service is not available.

- On OS/390, this reason code does not occur.

Completion Code: MQCC_FAILED

Programmer Response: Make the underlying service available.

MQRC_SIGNAL_OUTSTANDING
(2069, X'0815')

Explanation: An MQGET call was issued with either the MQGMO_SET_SIGNAL or MQGMO_WAIT option, but there is already a signal outstanding for the queue handle *Hobj*.

This reason code occurs only in the following environments: OS/390, Windows 95, Windows 98.

Completion Code: MQCC_FAILED

Programmer Response: Check the application logic. If it is necessary to set a signal or wait when there is a signal outstanding for the same queue, a different object handle must be used.

MQRC_SIGNAL_REQUEST_ACCEPTED
(2070, X'0816')

Explanation: An MQGET call was issued specifying MQGMO_SET_SIGNAL in the *GetMsgOpts* parameter, but no suitable message was available; the call returns immediately. The application can now wait for the signal to be delivered.

- On OS/390, the application should wait on the Event Control Block pointed to by the *Signal1* field.
- On Windows 95, Windows 98, the application should wait for the signal Windows message to be delivered.

This reason code occurs only in the following environments: OS/390, Windows 95, Windows 98.

Completion Code: MQCC_WARNING

Programmer Response: Wait for the signal; when it is delivered, check the signal to ensure that a message is now available. If it is, reissue the MQGET call.

- On OS/390, wait on the ECB pointed to by the *Signal1* field and, when it is posted, check it to ensure that a message is now available.
- On Windows 95, Windows 98, the application (thread) should continue executing its message loop.

MQRC_SIGNAL1_ERROR
(2099, X'0833')

Explanation: An MQGET call was issued, specifying MQGMO_SET_SIGNAL in the *GetMsgOpts* parameter, but the *Signal1* field is not valid.

- On OS/390, the address contained in the *Signal1* field is not valid, or points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- On Windows 95, Windows 98, the window handle in the *Signal1* field is not valid.

This reason code occurs only in the following environments: OS/390, Windows 95, Windows 98.

Completion Code: MQCC_FAILED

Programmer Response: Correct the setting of the *Signal1* field.

MQRC_SOURCE_BUFFER_ERROR
(2145, X'0861')

Explanation: On the MQXCNVC call, the *SourceBuffer* parameter pointer is not valid, or points to storage that cannot be accessed for the entire length specified by *SourceLength*. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

This reason code can also occur on the MQGET call when the MQGMO_CONVERT option is specified. In this case it indicates that the MQRC_SOURCE_BUFFER_ERROR reason was returned by an MQXCNVC call issued by the data conversion exit.

Completion Code: MQCC_WARNING or MQCC_FAILED

Programmer Response: Specify a valid buffer. If the reason code occurs on the MQGET call, check that the logic in the data-conversion exit is correct.

MQRC_SOURCE_CCSID_ERROR
(2111, X'083F')

Explanation: The coded character-set identifier from which character data is to be converted is not valid or not supported.

This can occur on the MQGET call when the MQGMO_CONVERT option is included in the *GetMsgOpts* parameter; the coded character-set identifier in error is the *CodedCharSetId* field in the message being retrieved. In this case, the message data is returned unconverted, the values of the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to those of the message returned, and the call completes with MQCC_WARNING.

This reason can also occur on the MQGET call when the message contains one or more MQ header structures (MQCIH, MQDLH, MQIIH, MQRMH), and the *CodedCharSetId* field in the message specifies a character set that does not have SBCS characters for the characters that are valid in queue names. MQ header structures containing such characters are not valid, and so the message is returned unconverted. The Unicode character set UCS-2 is an example of such a character set.

If the message consists of several parts, each of which is described by its own *CodedCharSetId* and *Encoding* fields (for example, a message with format name MQFMT_DEAD_LETTER_HEADER), some parts may be converted and other parts not converted. However, the values returned in the various *CodedCharSetId* and *Encoding* fields always correctly describe the relevant message data.

This reason can also occur on the MQXCNVC call; the coded character-set identifier in error is the *SourceCCSID* parameter. Either the *SourceCCSID* parameter specifies a value that is not valid or not supported, or the *SourceCCSID* parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code: MQCC_WARNING or MQCC_FAILED

Programmer Response: Check the character-set identifier that was specified when the message was put, or that was specified for the *SourceCCSID* parameter on the MQXCNVC call. If this is correct, check that it is one for which queue-manager conversion is supported. If queue-manager conversion is not supported for the specified character set, conversion must be carried out by the application.

MQRC_SOURCE_DECIMAL_ENC_ERROR (2113, X'0841')

Explanation: On an MQGET call with the MQGMO_CONVERT option included in the *GetMsgOpts* parameter, the *Encoding* value in the message being retrieved specifies a decimal encoding that is not recognized. The message data is returned unconverted, the values of the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to those of the message returned, and the call completes with MQCC_WARNING.

If the message consists of several parts, each of which is described by its own *CodedCharSetId* and *Encoding* fields (for example, a message with format name MQFMT_DEAD_LETTER_HEADER), some parts may be converted and other parts not converted. However, the values returned in the various *CodedCharSetId* and *Encoding* fields always correctly describe the relevant message data.

Completion Code: MQCC_WARNING

Programmer Response: Check the decimal encoding that was specified when the message was put. If this is correct, check that it is one for which queue-manager conversion is supported. If queue-manager conversion is not supported for the required decimal encoding, conversion must be carried out by the application.

MQRC_SOURCE_FLOAT_ENC_ERROR (2114, X'0842')

Explanation: On an MQGET call, with the MQGMO_CONVERT option included in the *GetMsgOpts* parameter, the *Encoding* value in the message being retrieved specifies a floating-point encoding that is not recognized. The message data is returned unconverted, the values of the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to those of the message returned, and the call completes with MQCC_WARNING.

If the message consists of several parts, each of which is described by its own *CodedCharSetId* and *Encoding* fields (for example, a message with format name MQFMT_DEAD_LETTER_HEADER), some parts may be converted and other parts not converted. However, the values returned in the various *CodedCharSetId* and *Encoding* fields always correctly describe the relevant message data.

Completion Code: MQCC_WARNING

Programmer Response: Check the floating-point encoding that was specified when the message was put. If this is correct, check that it is one for which queue-manager conversion is supported. If queue-manager conversion is not supported for the required floating-point encoding, conversion must be carried out by the application.

MQRC_SOURCE_INTEGER_ENC_ERROR (2112, X'0840')

Explanation: On an MQGET call, with the MQGMO_CONVERT option included in the *GetMsgOpts* parameter, the *Encoding* value in the message being retrieved specifies an integer encoding that is not recognized. The message data is returned unconverted, the values of the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to those of the message returned, and the call completes with MQCC_WARNING.

If the message consists of several parts, each of which is described by its own *CodedCharSetId* and *Encoding* fields (for example, a message with format name MQFMT_DEAD_LETTER_HEADER), some parts may be converted and other parts not converted. However, the values returned in the various *CodedCharSetId* and *Encoding* fields always correctly describe the relevant message data.

This reason code can also occur on the MQXCNVC call, when the *Options* parameter contains an unsupported MQDCC_SOURCE_* value, or when MQDCC_SOURCE_ENC_UNDEFINED is specified for a UCS2 code page.

Completion Code: MQCC_WARNING or MQCC_FAILED

Programmer Response: Check the integer encoding that was specified when the message was put. If this is correct, check that it is one for which queue-manager conversion is supported. If queue-manager conversion is not supported for the required integer encoding, conversion must be carried out by the application.

MQRC_SOURCE_LENGTH_ERROR
(2143, X'085F')

Explanation: On the MQXCNVC call, the *SourceLength* parameter specifies a length that is less than zero or not consistent with the string's character set or content (for example, the character set is a double-byte character set, but the length is not a multiple of two). This reason also occurs if the *SourceLength* parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

This reason code can also occur on the MQGET call when the MQGMO_CONVERT option is specified. In this case it indicates that the MQRC_SOURCE_LENGTH_ERROR reason was returned by an MQXCNVC call issued by the data conversion exit.

Completion Code: MQCC_WARNING or MQCC_FAILED

Programmer Response: Specify a length that is zero or greater. If the reason code occurs on the MQGET call, check that the logic in the data-conversion exit is correct.

MQRC_SRC_ENV_ERROR
(2261, X'08D5')

Explanation: This reason occurs when a channel exit that processes reference messages detects an error in the source environment data of a reference message header (MQRMH). One of the following is true:

- *SrcEnvLength* is less than zero.
- *SrcEnvLength* is greater than zero, but there is no source environment data.
- *SrcEnvLength* is greater than zero, but *SrcEnvOffset* is negative, zero, or less than the length of the fixed part of MQRMH.
- *SrcEnvLength* is greater than zero, but *SrcEnvOffset* plus *SrcEnvLength* is greater than *StrucLength*.

The exit returns this reason in the *Feedback* field of the MQCXP structure. If an exception report is requested, it is copied to the *Feedback* field of the MQMD associated with the report.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Specify the source environment data correctly.

MQRC_SRC_NAME_ERROR
(2262, X'08D6')

Explanation: This reason occurs when a channel exit that processes reference messages detects an error in the source name data of a reference message header (MQRMH). One of the following is true:

- *SrcNameLength* is less than zero.
- *SrcNameLength* is greater than zero, but there is no source name data.
- *SrcNameLength* is greater than zero, but *SrcNameOffset* is negative, zero, or less than the length of the fixed part of MQRMH.
- *SrcNameLength* is greater than zero, but *SrcNameOffset* plus *SrcNameLength* is greater than *StrucLength*.

The exit returns this reason in the *Feedback* field of the MQCXP structure. If an exception report is requested, it is copied to the *Feedback* field of the MQMD associated with the report.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Specify the source name data correctly.

MQRC_STOPPED_BY_CLUSTER_EXIT
(2188, X'088C')

Explanation: An MQOPEN, MQPUT, or MQPUT1 call was issued to open or put a message on a cluster queue, but the cluster workload exit rejected the call.

This reason code occurs in the following environments: AIX, HP-UX, OS/390, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Check the cluster workload exit to ensure that it has been written correctly. Determine why it rejected the call and correct the problem.

MQRC_STORAGE_CLASS_ERROR
(2105, X'0839')

Explanation: The MQPUT or MQPUT1 call was issued, but the storage-class object defined for the queue does not exist.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Create the storage-class object required by the queue, or modify the queue definition to use an existing storage class. The name of the storage-class object used by the queue is given by the *StorageClass* queue attribute.

MQRC_STORAGE_MEDIUM_FULL
(2192, X'0890')

Explanation: An MQI call was issued to operate on a queue, but the call failed because the external storage medium is full. One of the following applies:
- A page-set data set is full (nonshared queues only).
- A coupling-facility structure is full (shared queues only).

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Check which queues contain messages and look for applications that might be filling the queues unintentionally. Be aware that the queue that has caused the page set or coupling-facility structure to become full is not necessarily the queue referenced by the MQI call that returned MQRC_STORAGE_MEDIUM_FULL.

Check that all of the usual server applications are operating correctly and processing the messages on the queues.

If the applications and servers are operating correctly, increase the number of server applications to cope with the message load, or request the system programmer to increase the size of the page-set data sets.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'0817')

Explanation: The call failed because there is insufficient main storage available.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that active applications are behaving correctly, for example, that they are not looping unexpectedly. If no problems are found, make more main storage available.

- On OS/390, if no application problems are found, ask your systems programmer to increase the size of the region in which the queue manager runs.

MQRC_STRING_ERROR
(2307, X'0903')

Explanation: The *String* parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code: MQCC_FAILED

Programmer Response: Correct the parameter.

MQRC_STRING_LENGTH_ERROR
(2323, X'0913')

Explanation: The *StringLength* parameter is not valid. Either the parameter pointer is not valid, or it points to read-only storage. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code: MQCC_FAILED

Programmer Response: Correct the parameter.

MQRC_STRING_TRUNCATED
(2311, X'0907')

Explanation: The string returned by the call is too long to fit in the buffer provided. The string has been truncated to fit in the buffer.

Completion Code: MQCC_FAILED

Programmer Response: If the entire string is required, provide a larger buffer. On the mqInquireString call, the *StringLength* parameter is set by the call to indicate the size of the buffer required to accommodate the string without truncation.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'083D')

Explanation: On any call other than MQCONN or MQDISC, the API crossing exit suppressed the call.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Obey the rules for MQI calls that the exit enforces. To find out the rules, see the writer of the exit.

MQRC_SYNCPOINT_LIMIT_REACHED
(2024, X'07E8')

Explanation: An MQGET, MQPUT, or MQPUT1 call failed because it would have caused the number of uncommitted messages in the current unit of work to exceed the limit defined for the queue manager (see the *MaxUncommittedMsgs* queue-manager attribute). 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

- On Tandem NonStop Kernel, this reason code occurs when the maximum number of I/O operations in a single TM/MP transaction has been exceeded.

Completion Code: MQCC_FAILED

Programmer Response: Check whether the application is looping. If it is not, consider reducing the complexity of the application. Alternatively, increase the queue-manager limit for the maximum number of uncommitted messages within a unit of work.
- On OS/390, the limit for the maximum number of uncommitted messages can be changed by using the DEFINE MAXSMSGS command.
- On AS/400, the limit for the maximum number of uncommitted messages can be changed by using the CHGMQM command.
- On Tandem NonStop Kernel, the application should cancel the transaction and retry with a smaller number of operations in the unit of work. See the *MQSeries for Tandem NonStop Kernel System Management Guide* for more details.

MQRC_SYNCPOINT_NOT_AVAILABLE
(2072, X'0818')

Explanation: Either MQGMO_SYNCPOINT was specified on an MQGET call or MQPMO_SYNCPOINT was specified on an MQPUT or MQPUT1 call, but the local queue manager was unable to honor the request. If the queue manager does not support units of work, the *SyncPoint* queue-manager attribute will have the value MQSP_NOT_AVAILABLE.

This reason code can also occur on the MQGET, MQPUT, and MQPUT1 calls when an external unit-of-work coordinator is being used. If that coordinator requires an explicit call to start the unit of work, but the application has not issued that call prior to the MQGET, MQPUT, or MQPUT1 call, reason code MQRC_SYNCPOINT_NOT_AVAILABLE is returned.
- On AS/400, this reason codes means that AS/400 Commitment Control is not started, or is unavailable for use by the queue manager.
- On OS/390, this reason code does not occur.

Completion Code: MQCC_FAILED

Programmer Response: Remove the specification of MQGMO_SYNCPOINT or MQPMO_SYNCPOINT, as appropriate.
- On AS/400, ensure that Commitment Control has been started. If this reason code occurs after Commitment Control has been started, contact your systems programmer.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
(2315, X'090B')

Explanation: A call was issued to add a data item to a bag, modify the value of an existing data item in a bag, or retrieve a message into a bag, but the call failed because the bag is one that had been created by the system as a result of a previous mqExecute call. System bags cannot be modified by the application.

Completion Code: MQCC_FAILED

Programmer Response: Specify the handle of a bag created by the application, or remove the call.

MQRC_SYSTEM_BAG_NOT_DELETABLE
(2328, X'0918')

Explanation: An mqDeleteBag call was issued to delete a bag, but the call failed because the bag is one that had been created by the system as a result of a previous mqExecute call. System bags cannot be deleted by the application.

Completion Code: MQCC_FAILED

Programmer Response: Specify the handle of a bag created by the application, or remove the call.

MQRC_SYSTEM_ITEM_NOT_ALTERABLE
(2302, X'08FE')

Explanation: A call was issued to modify the value of a system data item in a bag (a data item with one of the MQIASY_* selectors), but the call failed because the data item is one that cannot be altered by the application.

Completion Code: MQCC_FAILED

Programmer Response: Specify the selector of a user-defined data item, or remove the call.

MQRC_SYSTEM_ITEM_NOT_DELETABLE
(2329, X'0919')

Explanation: A call was issued to delete a system data item from a bag (a data item with one of the MQIASY_* selectors), but the call failed because the data item is one that cannot be deleted by the application.

Completion Code: MQCC_FAILED

Programmer Response: Specify the selector of a user-defined data item, or remove the call.

MQRC_TARGET_BUFFER_ERROR
(2146, X'0862')

Explanation: On the MQXCNVC call, the *TargetBuffer* parameter pointer is not valid, or points to read-only storage, or to storage that cannot be accessed for the entire length specified by *TargetLength*. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

This reason code can also occur on the MQGET call when the MQGMO_CONVERT option is specified. In this case it indicates that the MQRC_TARGET_BUFFER_ERROR reason was returned

by an MQXCNVC call issued by the data conversion exit.

Completion Code: MQCC_WARNING or MQCC_FAILED

Programmer Response: Specify a valid buffer. If the reason code occurs on the MQGET call, check that the logic in the data-conversion exit is correct.

MQRC_TARGET_CCSID_ERROR
(2115, X'0843')

Explanation: The coded character-set identifier to which character data is to be converted is not valid or not supported.

This can occur on the MQGET call when the MQGMO_CONVERT option is included in the *GetMsgOpts* parameter; the coded character-set identifier in error is the *CodedCharSetId* field in the *MsgDesc* parameter. In this case, the message data is returned unconverted, the values of the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to those of the message returned, and the call completes with MQCC_WARNING.

This reason can also occur on the MQGET call when the message contains one or more MQ header structures (MQCIH, MQDLH, MQIIH, MQRMH), and the *CodedCharSetId* field in the *MsgDesc* parameter specifies a character set that does not have SBCS characters for the characters that are valid in queue names. The Unicode character set UCS-2 is an example of such a character set.

This reason can also occur on the MQXCNVC call; the coded character-set identifier in error is the *TargetCCSID* parameter. Either the *TargetCCSID* parameter specifies a value that is not valid or not supported, or the *TargetCCSID* parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)

Completion Code: MQCC_WARNING or MQCC_FAILED

Programmer Response: Check the character-set identifier that was specified for the *CodedCharSetId* field in the *MsgDesc* parameter on the MQGET call, or that was specified for the *SourceCCSID* parameter on the MQXCNVC call. If this is correct, check that it is one for which queue-manager conversion is supported. If queue-manager conversion is not supported for the specified character set, conversion must be carried out by the application.

MQRC_TARGET_DECIMAL_ENC_ERROR
(2117, X'0845')

Explanation: On an MQGET call with the MQGMO_CONVERT option included in the *GetMsgOpts* parameter, the *Encoding* value in the *MsgDesc* parameter specifies a decimal encoding that is not recognized. The message data is returned unconverted, the values of the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to those of the message returned, and the call completes with MQCC_WARNING.

Completion Code: MQCC_WARNING

Programmer Response: Check the decimal encoding that was specified. If this is correct, check that it is one for which queue-manager conversion is supported. If queue-manager conversion is not supported for the required decimal encoding, conversion must be carried out by the application.

MQRC_TARGET_FLOAT_ENC_ERROR
(2118, X'0846')

Explanation: On an MQGET call with the MQGMO_CONVERT option included in the *GetMsgOpts* parameter, the *Encoding* value in the *MsgDesc* parameter specifies a floating-point encoding that is not recognized. The message data is returned unconverted, the values of the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to those of the message returned, and the call completes with MQCC_WARNING.

Completion Code: MQCC_WARNING

Programmer Response: Check the floating-point encoding that was specified. If this is correct, check that it is one for which queue-manager conversion is supported. If queue-manager conversion is not supported for the required floating-point encoding, conversion must be carried out by the application.

MQRC_TARGET_INTEGER_ENC_ERROR
(2116, X'0844')

Explanation: On an MQGET call with the MQGMO_CONVERT option included in the *GetMsgOpts* parameter, the *Encoding* value in the *MsgDesc* parameter specifies an integer encoding that is not recognized. The message data is returned unconverted, the values of the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to those of the message being retrieved, and the call completes with MQCC_WARNING.

This reason code can also occur on the MQXCNVC call, when the *Options* parameter contains an unsupported MQDCC_TARGET_* value, or when MQDCC_TARGET_ENC_UNDEFINED is specified for a UCS2 code page.

Completion Code: MQCC_WARNING or MQCC_FAILED

Programmer Response: Check the integer encoding that was specified. If this is correct, check that it is one for which queue-manager conversion is supported. If queue-manager conversion is not supported for the

required integer encoding, conversion must be carried out by the application.

MQRC_TARGET_LENGTH_ERROR
(2144, X'0860')

Explanation: On the MQXCNVC call, the *TargetLength* parameter is not valid for one of the following reasons:

- *TargetLength* is less than zero.
- The *TargetLength* parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The MQDCC_FILL_TARGET_BUFFER option is specified, but the value of *TargetLength* is such that the target buffer cannot be filled completely with valid characters. 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.

This reason code can also occur on the MQGET call when the MQGMO_CONVERT option is specified. In this case it indicates that the MQRC_TARGET_LENGTH_ERROR reason was returned by an MQXCNVC call issued by the data conversion exit.

Completion Code: MQCC_WARNING or MQCC_FAILED

Programmer Response: Specify a length that is zero or greater. If the MQDCC_FILL_TARGET_BUFFER option is specified, and *TargetCCSID* is a pure DBCS character set, ensure that *TargetLength* specifies a length that is a multiple of two.

If the reason code occurs on the MQGET call, check that the logic in the data-conversion exit is correct.

MQRC_TERMINATION_FAILED
(2287, X'08EF')

Explanation: This reason should be returned by an installable service component when the component is unable to complete termination successfully.

- On OS/390, this reason code does not occur.

Completion Code: MQCC_FAILED

Programmer Response: Correct the error and retry the operation.

MQRC_TM_ERROR
(2265, X'08D9')

Explanation: On an MQPUT or MQPUT1 call, the trigger message structure MQTM in the message data is not valid.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Correct the definition of the MQTM structure. Ensure that the fields are set correctly.

MQRC_TMC_ERROR
(2191, X'088F')

Explanation: On an MQPUT or MQPUT1 call, the character trigger message structure MQTMC or MQTMC2 in the message data is not valid.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Correct the definition of the MQTMC or MQTMC2 structure. Ensure that the fields are set correctly.

MQRC_TRIGGER_CONTROL_ERROR
(2075, X'081B')

Explanation: On an MQSET call, the value specified for the MQIA_TRIGGER_CONTROL attribute selector is not valid.

Completion Code: MQCC_FAILED

Programmer Response: Specify a valid value. See "Chapter 39. Attributes for queues" on page 427.

MQRC_TRIGGER_DEPTH_ERROR
(2076, X'081C')

Explanation: On an MQSET call, the value specified for the MQIA_TRIGGER_DEPTH attribute selector is not valid.

Completion Code: MQCC_FAILED

Programmer Response: Specify a value that is greater than zero. See "Chapter 39. Attributes for queues" on page 427.

MQRC_TRIGGER_MSG_PRIORITY_ERR
(2077, X'081D')

Explanation: On an MQSET call, the value specified for the MQIA_TRIGGER_MSG_PRIORITY attribute selector is not valid.

Completion Code: MQCC_FAILED

Programmer Response: Specify a value in the range zero through the value of *MaxPriority* queue-manager attribute. See "Chapter 39. Attributes for queues" on page 427.

MQRC_TRIGGER_TYPE_ERROR
(2078, X'081E')

Explanation: On an MQSET call, the value specified for the MQIA_TRIGGER_TYPE attribute selector is not valid.

Completion Code: MQCC_FAILED

Programmer Response: Specify a valid value. See "Chapter 39. Attributes for queues" on page 427.

MQRC_TRUNCATED_MSG_ACCEPTED
(2079, X'081F')

Explanation: On an MQGET call, the message length was too large to fit into the supplied buffer. The MQGMO_ACCEPT_TRUNCATED_MSG option was specified, so the call completes. The message is removed from the queue (subject to unit-of-work considerations), or, if this was a browse operation, the browse cursor is advanced to this message.

The *DataLength* parameter is set to the length of the message before truncation, the *Buffer* parameter contains as much of the message as fits, and the MQMD structure is filled in.

Completion Code: MQCC_WARNING

Programmer Response: None, because the application expected this situation.

MQRC_TRUNCATED_MSG_FAILED
(2080, X'0820')

Explanation: On an MQGET call, the message length was too large to fit into the supplied buffer. The MQGMO_ACCEPT_TRUNCATED_MSG option was *not* specified, so the message has not been removed from the queue. If this was a browse operation, the browse cursor remains where it was before this call, but if MQGMO_BROWSE_FIRST was specified, the browse cursor is positioned logically before the highest-priority message on the queue.

The *DataLength* field is set to the length of the message before truncation, the *Buffer* parameter contains as much of the message as fits, and the MQMD structure is filled in.

Completion Code: MQCC_WARNING

Programmer Response: Supply a buffer that is at least as large as *DataLength*, or specify MQGMO_ACCEPT_TRUNCATED_MSG if not all of the message data is required.

MQRC_UCS2_CONVERSION_ERROR
(2341, X'0925')

Explanation: This reason code is returned by the Java MQQueueManager constructor when a required character-set conversion is not available. The conversion required is between the UCS-2 Unicode character set and the queue-manager's character set. IBM-500 is used for the queue-manager's character set if no specific value is available.

This reason code occurs in the following environment: MQSeries classes for Java on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the relevant Unicode conversion tables are installed, and that they are available to the OS/390 Language Environment. The conversion tables should be installed as part of the OS/390 C/C++ optional feature. Refer to the *OS/390 C/C++ Programming Guide* for more information about enabling UCS-2 conversions.

MQRC_UNEXPECTED_ERROR
(2195, X'0893')

Explanation: The call was rejected because an unexpected error occurred.

Completion Code: MQCC_FAILED

Programmer Response: Check the application's parameter list to ensure, for example, that the correct number of parameters was passed, and that data pointers and storage keys are valid. If the problem cannot be resolved, contact your system programmer.

- On OS/390, check whether any information has been displayed on the console. If this error occurs on an MQCONN or MQCONNX call, check that the subsystem named is an active MQ subsystem. In particular, check that it is not a DB2™ subsystem. If the problem cannot be resolved, rerun the application with a CSQSNAP DD card (if you have not already got a dump) and send the resulting dump to IBM.
- On OS/2 and AS/400, consult the FFST record to obtain more detail about the problem.
- On Compaq (DIGITAL) OpenVMS, Tandem NonStop Kernel, and UNIX systems, consult the FDC file to obtain more detail about the problem.

MQRC_UNIT_OF_WORK_NOT_STARTED
(2232, X'08B8')

Explanation: An MQGET, MQPUT or MQPUT1 call was issued to get or put a message within a unit of work, but no TM/MP transaction had been started. If MQGMO_NO_SYNCPOINT is not specified on MQGET, or MQPMO_NO_SYNCPOINT is not specified on MQPUT or MQPUT1 (the default), the call requires a unit of work.

Completion Code: MQCC_FAILED

Programmer Response: Ensure a TM/MP transaction is available, or issue the MQGET call with the MQGMO_NO_SYNCPOINT option, or the MQPUT or MQPUT1 call with the MQPMO_NO_SYNCPOINT

option, which will cause a transaction to be started automatically.

MQRC_UNKNOWN_ALIAS_BASE_Q
(2082, X'0822')

Explanation: An MQOPEN or MQPUT1 call was issued specifying an alias queue as the target, but the *BaseQName* in the alias queue attributes is not recognized as a queue name.

This reason code can also occur when *BaseQName* is the name of a cluster queue that cannot be resolved successfully.

Completion Code: MQCC_FAILED

Programmer Response: Correct the queue definitions.

MQRC_UNKNOWN_AUTH_ENTITY
(2293, X'08F5')

Explanation: This reason should be returned by the authority installable service component when the name specified by the *AuthEntityName* parameter is not recognized.

- On OS/390, this reason code does not occur.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the entity is defined.

MQRC_UNKNOWN_DEF_XMIT_Q
(2197, X'0895')

Explanation: An MQOPEN or MQPUT1 call was issued specifying a remote queue as the destination. If a local definition of the remote queue was specified, or if a queue-manager alias is being resolved, the *XmitQName* attribute in the local definition is blank.

Because there is no queue defined with the same name as the destination queue manager, the queue manager has attempted to use the default transmission queue. However, the name defined by the *DefXmitQName* queue-manager attribute is not the name of a locally-defined queue.

Completion Code: MQCC_FAILED

Programmer Response: Correct the queue definitions, or the queue-manager attribute. See the *MQSeries Application Programming Guide* for more information.

MQRC_UNKNOWN_ENTITY
(2292, X'08F4')

Explanation: This reason should be returned by the authority installable service component when the name specified by the *EntityName* parameter is not recognized.

- On OS/390, this reason code does not occur.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the entity is defined.

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'0825')

Explanation: An MQOPEN or MQPUT1 call was issued, but the object identified by the *ObjectName* and *ObjectQMgrName* fields in the object descriptor MQOD cannot be found. One of the following applies:

- The *ObjectQMgrName* field is one of the following:
  - Blank
  - The name of the local queue manager
  - The name of a local definition of a remote queue (a queue-manager alias) in which the *RemoteQMgrName* attribute is the name of the local queue manager

  but no object with the specified *ObjectName* and *ObjectType* exists on the local queue manager.
- The object being opened is a cluster queue that is hosted on a remote queue manager, but the local queue manager does not have a defined route to the remote queue manager.
- The object being opened is a queue definition that has QSGDISP(GROUP). Such definitions cannot be used with the MQOPEN and MQPUT1 calls.

Completion Code: MQCC_FAILED

Programmer Response: Specify a valid object name. Ensure that the name is padded to the right with blanks if necessary. If this is correct, check the queue definitions.

MQRC_UNKNOWN_OBJECT_Q_MGR
(2086, X'0826')

Explanation: On an MQOPEN or MQPUT1 call, the *ObjectQMgrName* field in the object descriptor MQOD does not satisfy the naming rules for objects. For more information, see the *MQSeries Application Programming Guide*.

This reason also occurs if the *ObjectType* field in the object descriptor has the value MQOT_Q_MGR, and the *ObjectQMgrName* field is not blank, but the name specified is not the name of the local queue manager.

Completion Code: MQCC_FAILED

Programmer Response: Specify a valid queue manager name. To refer to the local queue manager, a name consisting entirely of blanks or beginning with a null character can be used. Ensure that the name is padded to the right with blanks or terminated with a null character if necessary.

MQRC_UNKNOWN_Q_NAME
(2288, X'08F0')

Explanation: This reason should be returned by the MQZ_LOOKUP_NAME installable service component when the name specified for the *QName* parameter is not recognized.

- On OS/390, this reason code does not occur.

Completion Code: MQCC_FAILED

Programmer Response: None. See the *MQSeries Programmable System Management* book for information about installable services.

MQRC_UNKNOWN_REF_OBJECT
(2294, X'08F6')

Explanation: This reason should be returned by the MQZ_COPY_ALL_AUTHORITY installable service component when the name specified by the *RefObjectName* parameter is not recognized.

- On OS/390, this reason code does not occur.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the reference object is defined. See the *MQSeries Programmable System Management* book for information about installable services.

MQRC_UNKNOWN_REMOTE_Q_MGR
(2087, X'0827')

Explanation: On an MQOPEN or MQPUT1 call, an error occurred with the queue-name resolution, for one of the following reasons:

- *ObjectQMgrName* is blank or the name of the local queue manager, *ObjectName* is the name of a local definition of a remote queue (or an alias to one), and one of the following is true:
  - *RemoteQMgrName* is blank or the name of the local queue manager. Note that this error occurs even if *XmitQName* is not blank.
  - *XmitQName* is blank, but there is no transmission queue defined with the name of *RemoteQMgrName*, and the *DefXmitQName* queue-manager attribute is blank.
  - *RemoteQMgrName* and *RemoteQName* specify a cluster queue that cannot be resolved successfully, and the *DefXmitQName* queue-manager attribute is blank.
- *ObjectQMgrName* is the name of a local definition of a remote queue (containing a queue-manager alias definition), and one of the following is true:
  - *RemoteQName* is not blank.
  - *XmitQName* is blank, but there is no transmission queue defined with the name of *RemoteQMgrName*, and the *DefXmitQName* queue-manager attribute is blank.
- *ObjectQMgrName* is not:
  - Blank
  - The name of the local queue manager
  - The name of a transmission queue
  - The name of a queue-manager alias definition (that is, a local definition of a remote queue with a blank *RemoteQName*)

  but the *DefXmitQName* queue-manager attribute is blank.
- *ObjectQMgrName* is the name of a model queue.
- The queue name is resolved through a cell directory. However, there is no queue defined with the same name as the remote queue manager name obtained from the cell directory, and the *DefXmitQName* queue-manager attribute is blank.

Completion Code: MQCC_FAILED

Programmer Response: Check the values specified for *ObjectQMgrName* and *ObjectName*. If these are correct, check the queue definitions.

MQRC_UNKNOWN_REPORT_OPTION
(2104, X'0838')

Explanation: An MQPUT or MQPUT1 call was issued, but the *Report* field in the message descriptor MQMD contains one or more options that are not recognized by the local queue manager. The options are accepted.

The options that cause this reason code to be returned depend on the destination of the message; see "Appendix E. Report options and message flags" on page 593 for more details.

Completion Code: MQCC_WARNING

Programmer Response: If this reason code is expected, no corrective action is required. If this reason code is not expected, do the following:

- Ensure that the *Report* field in the message descriptor is initialized with a value when the message descriptor is declared, or is assigned a value prior to the MQPUT or MQPUT1 call.
- Ensure that the report options specified are ones that are documented in this book; see the *Report* field described in "Chapter 9. MQMD - Message descriptor" on page 125 for valid report options. Remove any report options that are not documented in this book.
- If multiple report options are being set by adding the individual report options together, ensure that the same report option is not added twice.
- Check that conflicting report options are not specified. For example, do not add both MQRO_EXCEPTION and MQRO_EXCEPTION_WITH_DATA to the *Report* field; only one of these can be specified.

MQRC_UNKNOWN_XMIT_Q
(2196, X'0894')

Explanation: On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager. The *ObjectName* or the *ObjectQMgrName* in the object descriptor specifies the name of a local definition of a remote queue (in the latter case queue-manager aliasing is being used), but the *XmitQName* attribute of the definition is not blank and not the name of a locally-defined queue.

Completion Code: MQCC_FAILED

Programmer Response: Check the values specified for *ObjectName* and *ObjectQMgrName*. If these are correct, check the queue definitions. For more information on transmission queues, see the *MQSeries Application Programming Guide*.

MQRC_UOW_CANCELED
(2297, X'08F9')

Explanation: An MQI call was issued, but the unit of work (TM/MP transaction) being used for the MQ operation had been canceled. This may have been done by TM/MP itself (for example, due to the transaction running for too long, or exceeding audit trail sizes), or by the application program issuing an ABORT_TRANSACTION. All updates performed to MQSeries resources are backed out.

Completion Code: MQCC_FAILED

Programmer Response: Refer to the operating system's *Transaction Management Operations Guide* to determine how the Transaction Manager can be tuned to avoid the problem of system limits being exceeded.

MQRC_UOW_ENLISTMENT_ERROR
(2354, X'0932')

Explanation: This reason code can occur for a variety of reasons. The most likely reason is that an object created by a DTC transaction does not issue a transactional MQI call until after the DTC transaction has timed out. (If the DTC transaction times out after a transactional MQI call has been issued, reason code MQRC_HANDLE_IN_USE_FOR_UOW is returned by the failing MQI call.)

Another cause of MQRC_UOW_ENLISTMENT_ERROR is incorrect installation; Windows NT Service pack must be installed after the Windows NT Option pack.

This reason code occurs only on Windows NT.

Completion Code: MQCC_FAILED

Programmer Response: Check the DTC "Transaction timeout" value. If necessary, verify the NT installation order.

MQRC_UOW_IN_PROGRESS
(2128, X'0850')

Explanation: An MQBEGIN call was issued to start a unit of work coordinated by the queue manager, but a unit of work is already in existence for the connection handle specified. This may be a global unit of work started by a previous MQBEGIN call, or a unit of work that is local to the queue manager or one of the cooperating resource managers. No more than one unit of work can exist concurrently for a connection handle.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT.

Completion Code: MQCC_FAILED

Programmer Response: Review the application logic to determine why there is a unit of work already in existence. Move the MQBEGIN call to the appropriate place in the application.

MQRC_UOW_MIX_NOT_SUPPORTED
(2355, X'0933')

Explanation: The mixture of calls used by the application to perform operations within a unit of work is not supported. In particular, it is not possible to mix within the same process a local unit of work coordinated by the queue manager with a global unit of work coordinated by DTC (Distributed Transaction Coordinator).

An application may cause this mixture to arise if some objects in a package are coordinated by DTC and others are not. It can also occur if transactional MQI calls from an MTS client are mixed with transactional MQI calls from a library package transactional MTS object.

No problem arises if all transactional MQI calls originate from transactional MTS objects, or all transactional MQI calls originate from nontransactional MTS objects. But when a mixture of styles is used, the first style used fixes the style for the unit of work, and subsequent attempts to use the other style within the process fail with reason code MQRC_UOW_MIX_NOT_SUPPORTED.

When an application is run twice, scheduling factors in the operating system mean that it is possible for the queue-manager-coordinated transactional calls to fail in one run, and for the DTC-coordinated transactional calls to fail in the other run.

This reason code occurs only on Windows NT.

Completion Code: MQCC_FAILED

Programmer Response: Check that the "MTS Transaction Support" attribute defined for the object's class is set correctly. If necessary, modify the application so that objects executing within different units of work do not try to use the same connection handle.

MQRC_UOW_NOT_AVAILABLE
(2255, X'08CF')

Explanation: An MQGET, MQPUT, or MQPUT1 call was issued to get or put a message outside a unit of work, but the options specified on the call required the queue manager to process the call within a unit of work. Because there is already a user-defined unit of work in existence, the queue manager was unable to create a temporary unit of work for the duration of the call.

This reason occurs in the following circumstances:
- On an MQGET call, when the MQGMO_COMPLETE_MSG option is specified in MQGMO and the logical message to be retrieved is persistent and consists of two or more segments.
- On an MQPUT or MQPUT1 call, when the MQMF_SEGMENTATION_ALLOWED flag is specified in MQMD and the message requires segmentation.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Issue the MQGET, MQPUT, or MQPUT1 call inside the user-defined unit of work. Alternatively, for the MQPUT or MQPUT1 call, reduce the size of the message so that it does not require segmentation by the queue manager.

MQRC_USER_ID_NOT_AVAILABLE
(2291, X'08F3')

Explanation: This reason should be returned by the MQZ_FIND_USERID installable service component when the user ID cannot be determined.

- On OS/390, this reason code does not occur.

Completion Code: MQCC_FAILED

Programmer Response: None. See the *MQSeries Programmable System Management* book for information about installable services.

MQRC_WAIT_INTERVAL_ERROR
(2090, X'082A')

Explanation: On the MQGET call, the value specified for the *WaitInterval* field in the *GetMsgOpts* parameter is not valid.

Completion Code: MQCC_FAILED

Programmer Response: Specify a value greater than or equal to zero, or the special value MQWI_UNLIMITED if an indefinite wait is required.

MQRC_WIH_ERROR
(2333, X'091D')

Explanation: An MQPUT or MQPUT1 call was issued to put a message on a queue whose *IndexType* attribute had the value MQIT_MSG_TOKEN, but the message data did not begin with a valid MQWIH structure.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Modify the application to ensure that it places a valid MQWIH structure at the start of the message data.

MQRC_WRONG_GMO_VERSION
(2256, X'08D0')

Explanation: An MQGET call was issued specifying options that required an MQGMO with a version number not less than MQGMO_VERSION_2, but the MQGMO supplied did not satisfy this condition.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Modify the application to pass a version-2 MQGMO. Check the application logic to ensure that the *Version* field in MQGMO has been set to MQGMO_VERSION_2. Alternatively, remove the option that requires the version-2 MQGMO.

MQRC_WRONG_MD_VERSION
(2257, X'08D1')

Explanation: An MQGET, MQPUT, or MQPUT1 call was issued specifying options that required an MQMD with a version number not less than MQMD_VERSION_2, but the MQMD supplied did not satisfy this condition.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Modify the application to pass a version-2 MQMD. Check the application logic to ensure that the *Version* field in MQMD has been set to MQMD_VERSION_2. Alternatively, remove the option that requires the version-2 MQMD.

MQRC_WXP_ERROR
(2356, X'0934')

Explanation: An MQXCLWLN call was issued from a cluster workload exit to obtain the address of the next record in the chain, but the workload exit parameter structure *ExitParms* is not valid, for one of the following reasons:

- The parameter pointer is not valid. (It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.)
- The *StrucId* field is not MQWXP_STRUC_ID.
- The *Version* field is not MQWXP_VERSION_2.

Completion Code: MQCC_FAILED

Programmer Response: Ensure that the parameter specified for *ExitParms* is the MQWXP structure that was passed to the exit when the exit was invoked.

MQRC_XMIT_Q_TYPE_ERROR
(2091, X'082B')

Explanation: On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager. The *ObjectName* or *ObjectQMgrName* field in the object descriptor specifies the name of a local definition of a remote queue but one of the following applies to the *XmitQName* attribute of the definition:

- *XmitQName* is not blank, but specifies a queue that is not a local queue
- *XmitQName* is blank, but *RemoteQMgrName* specifies a queue that is not a local queue

This reason also occurs if the queue name is resolved through a cell directory, and the remote queue manager name obtained from the cell directory is the name of a queue, but this is not a local queue.

Completion Code: MQCC_FAILED

Programmer Response: Check the values specified for *ObjectName* and *ObjectQMgrName*. If these are correct, check the queue definitions. For more information on transmission queues, see the *MQSeries Application Programming Guide*.

MQRC_XMIT_Q_USAGE_ERROR
(2092, X'082C')

Explanation: On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager, but one of the following occurred:

- *ObjectQMgrName* specifies the name of a local queue, but it does not have a *Usage* attribute of MQUS_TRANSMISSION.
- The *ObjectName* or *ObjectQMgrName* field in the object descriptor specifies the name of a local definition of a remote queue but one of the following applies to the *XmitQName* attribute of the definition:
  - *XmitQName* is not blank, but specifies a queue that does not have a *Usage* attribute of MQUS_TRANSMISSION
  - *XmitQName* is blank, but *RemoteQMgrName* specifies a queue that does not have a *Usage* attribute of MQUS_TRANSMISSION
- The queue name is resolved through a cell directory, and the remote queue manager name obtained from the cell directory is the name of a local queue, but it does not have a *Usage* attribute of MQUS_TRANSMISSION.

Completion Code: MQCC_FAILED

Programmer Response: Check the values specified for *ObjectName* and *ObjectQMgrName*. If these are correct, check the queue definitions. For more information on transmission queues, see the *MQSeries Application Programming Guide*.

MQRC_XQH_ERROR
(2260, X'08D4')

Explanation: On an MQPUT or MQPUT1 call, the transmission queue header structure MQXQH in the message data is not valid.

This reason code occurs in the following environments: AIX, HP-UX, OS/2, AS/400, Sun Solaris, Windows NT, plus MQSeries clients connected to these systems.

Completion Code: MQCC_FAILED

Programmer Response: Correct the definition of the MQXQH structure. Ensure that the fields are set correctly.

MQRC_XWAIT_CANCELED
(2107, X'083B')

Explanation: An MQXWAIT call was issued, but the call has been canceled because a STOP CHINIT command has been issued (or the queue manager has been stopped, which causes the same effect). Refer to the *MQSeries Intercommunication* book for details of the MQXWAIT call.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Tidy up and terminate.

MQRC_XWAIT_ERROR
(2108, X'083C')

Explanation: An MQXWAIT call was issued, but the invocation was not valid for one of the following reasons:

- The wait descriptor MQXWD contains data that is not valid.
- The linkage stack level is not valid.
- The addressing mode is not valid.
- There are too many wait events outstanding.

This reason code occurs only on OS/390.

Completion Code: MQCC_FAILED

Programmer Response: Obey the rules for using the MQXWAIT call. Refer to the *MQSeries Intercommunication* book for details of this call.

Appendix B. MQSeries constants

This chapter specifies the values of all of the named constants that are mentioned in this book.

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 "MQ*xxxx*_", where *xxxx* represents a string of 0 through 4 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 33. MQINQ - Inquire about object attributes" on page 363 and "Chapter 37. MQSET - Set object attributes" on page 415.

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'
MQ_FORMAT_LENGTH	8	X'00000008'
MQ_FUNCTION_LENGTH	4	X'00000004'
MQ_GROUP_ID_LENGTH	24	X'00000018'

© Copyright IBM Corp. 1994, 2000

MQ_LTERM_OVERRIDE_LENGTH	8	X'00000008'
MQ_MFS_MAP_NAME_LENGTH	8	X'00000008'
MQ_MSG_HEADER_LENGTH	4000	X'00000FA0'
MQ_MSG_ID_LENGTH	24	X'00000018'
MQ_MSG_TOKEN_LENGTH	16	X'00000010'
MQ_NAMELIST_DESC_LENGTH	64	X'00000040'
MQ_NAMELIST_NAME_LENGTH	48	X'00000030'
MQ_OBJECT_INSTANCE_ID_LENGTH	24	X'00000018'
MQ_OBJECT_NAME_LENGTH	48	X'00000030'
MQ_PROCESS_APPL_ID_LENGTH	256	X'00000100'
MQ_PROCESS_DESC_LENGTH	64	X'00000040'
MQ_PROCESS_ENV_DATA_LENGTH	128	X'00000080'
MQ_PROCESS_NAME_LENGTH	48	X'00000030'
MQ_PROCESS_USER_DATA_LENGTH	128	X'00000080'
MQ_PUT_APPL_NAME_LENGTH	28	X'0000001C'
MQ_PUT_DATE_LENGTH	8	X'00000008'
MQ_PUT_TIME_LENGTH	8	X'00000008'
MQ_Q_DESC_LENGTH	64	X'00000040'
MQ_Q_MGR_DESC_LENGTH	64	X'00000040'
MQ_Q_MGR_IDENTIFIER_LENGTH	48	X'00000030'
MQ_Q_MGR_NAME_LENGTH	48	X'00000030'
MQ_Q_NAME_LENGTH	48	X'00000030'
MQ_QSG_NAME_LENGTH	4	X'00000004'
MQ_REMOTE_SYS_ID_LENGTH	4	X'00000004'
MQ_SERVICE_NAME_LENGTH	32	X'00000020'
MQ_SERVICE_STEP_LENGTH	8	X'00000008'
MQ_START_CODE_LENGTH	4	X'00000004'
MQ_STORAGE_CLASS_LENGTH	8	X'00000008'
MQ_TIME_LENGTH	8	X'00000008'
MQ_TRAN_INSTANCE_ID_LENGTH	16	X'00000010'
MQ_TRANSACTION_ID_LENGTH	4	X'00000004'
MQ_TRIGGER_DATA_LENGTH	64	X'00000040'
MQ_USER_ID_LENGTH	12	X'0000000C'

MQACT_* (Accounting token)

See the *AccountingToken* field described in "Chapter 9. MQMD - Message descriptor" on page 125.

MQACT_NONE	X'00...00' (32 nulls)

For the C programming language, the following array version is also defined:

MQACT_NONE_ARRAY	'\0','\0',...'\0','\0'

MQACTT_* (Accounting token type)

See the *AccountingToken* field described in "Chapter 9. MQMD - Message descriptor" on page 125.

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'

MQAT_* (Application type)

See the *PutApplType* field described in "Chapter 9. MQMD - Message descriptor" on page 125, and the *ApplType* attribute described in "Chapter 41. Attributes for process definitions" on page 463.

MQAT_DEFAULT	(variable)	
MQAT_UNKNOWN	-1	X'FFFFFFFF'
MQAT_NO_CONTEXT	0	X'00000000'
MQAT_CICS	1	X'00000001'
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_MVS	2	X'00000002'
MQAT_OS390	2	X'00000002'
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_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_USER_FIRST	65536	X'00010000'
MQAT_QMGR	7	X'00000007'
MQAT_OS400	8	X'00000008'
MQAT_WINDOWS	9	X'00000009'
MQAT_USER_LAST	999999999	X'3B9AC9FF'

MQBND_* (Binding)

See the *DefBind* attribute described in "Chapter 39. Attributes for queues" on page 427.

MQBND_BIND_ON_OPEN	0	X'00000000'
MQBND_BIND_NOT_FIXED	1	X'00000001'

MQBO_* (Begin options)

See the *Options* field described in "Chapter 2. MQBO - Begin options" on page 29.

MQBO_NONE	0	X'00000000'

MQBO_* (Begin options structure identifier)

See the *StrucId* field described in "Chapter 2. MQBO - Begin options" on page 29.

MQBO_STRUC_ID	'BObb'

For the C programming language, the following array version is also defined:

MQBO_STRUC_ID_ARRAY	'B','O','b','b'

MQBO_* (Begin options version)

See the *Version* field described in "Chapter 2. MQBO - Begin options" on page 29.

MQBO_VERSION_1	1	X'00000001'
MQBO_CURRENT_VERSION	1	X'00000001'

MQCA_* (Character attribute selector)

See the *Selectors* parameter described in "Chapter 33. MQINQ - Inquire about object attributes" on page 363 and "Chapter 37. MQSET - Set object attributes" on page 415.

MQCA_LAST_USED	(variable)	
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'
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'

MQCADSD_* (CICS header ADS descriptor)

See the *ADSDescriptor* field described in "Chapter 3. MQCIH - CICS information header" on page 33.

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_UNKNOWN	-1	X'FFFFFFFF'
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 9. MQMD - Message descriptor" on page 125 and in other structures.

MQCCSI_EMBEDDED	-1	X'FFFFFFFF'
MQCCSI_INHERIT	-2	X'FFFFFFFE'
MQCCSI_Q_MGR	0	X'00000000'
MQCCSI_DEFAULT	0	X'00000000'
MQCCSI_UNDEFINED	0	X'00000000'

MQCCT_* (CICS header conversational task)

See the *ConversationalTask* field described in "Chapter 3. MQCIH - CICS information header" on page 33.

MQCCT_NO	0	X'00000000'
MQCCT_YES	1	X'00000001'

MQCFAC_* (CICS header facility)

See the *Facility* field described in "Chapter 3. MQCIH - CICS information header" on page 33.

MQCFAC_NONE	X'00...00' (8 nulls)

For the C programming language, the following array version is also defined:

MQCFAC_NONE_ARRAY	'\0','\0',...'\0','\0'

MQCFUNC_* (CICS header function name)

See the *Function* field described in "Chapter 3. MQCIH - CICS information header" on page 33.

MQCFUNC_MQCONN	'CONN'
MQCFUNC_MQGET	'GETb'
MQCFUNC_MQINQ	'INQb'
MQCFUNC_MQOPEN	'OPEN'
MQCFUNC_MQPUT	'PUTb'
MQCFUNC_MQPUT1	'PUT1'
MQCFUNC_NONE	'bbbb'

For the C programming language, the following array versions are also defined:

MQCFUNC_MQCONN_ARRAY	'C','O','N','N'
MQCFUNC_MQGET_ARRAY	'G','E','T','b'
MQCFUNC_MQINQ_ARRAY	'I','N','Q','b'
MQCFUNC_MQOPEN_ARRAY	'O','P','E','N'
MQCFUNC_MQPUT_ARRAY	'P','U','T','b'
MQCFUNC_MQPUT1_ARRAY	'P','U','T','1'
MQCFUNC_NONE_ARRAY	'b','b','b','b'

MQCGWI_* (CICS header get-wait interval)

See the *GetWaitInterval* field described in "Chapter 3. MQCIH - CICS information header" on page 33.

MQCGWI_DEFAULT	-2	X'FFFFFFFE'

MQCI_* (Correlation identifier)

See the *CorrelId* field described in "Chapter 9. MQMD - Message descriptor" on page 125.

MQCI_NONE	X'00...00' (24 nulls)
MQCI_NEW_SESSION	X'414D51214E45575F534553...'

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',...

MQCIH_* (CICS header flags)

See the *Flags* field described in "Chapter 3. MQCIH - CICS information header" on page 33.

MQCIH_NONE	0	X'00000000'

MQCIH_* (CICS header length)

See the *StrucLength* field described in "Chapter 3. MQCIH - CICS information header" on page 33.

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 3. MQCIH - CICS information header" on page 33.

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 3. MQCIH - CICS information header" on page 33.

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 3. MQCIH - CICS information header" on page 33.

MQCLT_PROGRAM	1	X'00000001'
MQCLT_TRANSACTION	2	X'00000002'

MQCMDL_* (Command level)

See the *CommandLevel* attribute described in "Chapter 42. Attributes for the queue manager" on page 469.

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'

MQCNO_* (Connect options)

See the *Options* field described in "Chapter 4. MQCNO - Connect options" on page 51.

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_NONE	0	X'00000000'

MQCNO_* (Connect options structure identifier)

See the *StrucId* field described in "Chapter 4. MQCNO - Connect options" on page 51.

MQCNO_STRUC_ID	'CNOb'

For the C programming language, the following array version is also defined:

MQCNO_STRUC_ID_ARRAY	'C','N','O','b'

MQCNO_* (Connect options version)

See the *Version* field described in "Chapter 4. MQCNO - Connect options" on page 51.

MQCNO_VERSION_1	1	X'00000001'
MQCNO_VERSION_2	2	X'00000002'
MQCNO_VERSION_3	3	X'00000003'
MQCNO_CURRENT_VERSION	(variable)	

MQCO_* (Close options)

See the *Options* parameter described in "Chapter 27. MQCLOSE - Close object" on page 319.

MQCO_NONE	0	X'00000000'
MQCO_DELETE	1	X'00000001'
MQCO_DELETE_PURGE	2	X'00000002'

MQCODL_* (CICS header output data length)

See the *OutputDataLength* field described in "Chapter 3. MQCIH - CICS information header" on page 33.

MQCODL_AS_INPUT	-1	X'FFFFFFFF'

MQCRC_* (CICS header return code)

See the *ReturnCode* field described in "Chapter 3. MQCIH - CICS information header" on page 33.

MQCRC_OK	0	X'00000000'
MQCRC_CICS_EXEC_ERROR	1	X'00000001'
MQCRC_MQ_API_ERROR	2	X'00000002'
MQCRC_BRIDGE_ERROR	3	X'00000003'
MQCRC_BRIDGE_ABEND	4	X'00000004'
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 3. MQCIH - CICS information header" on page 33.

MQCSC_START	'Sbbb'
MQCSC_STARTDATA	'SDbb'
MQCSC_TERMINPUT	'TDbb'
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 4. MQCNO - Connect options" on page 51.

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 3. MQCIH - CICS information header" on page 33.

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 3. MQCIH - CICS information header" on page 33.

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 "MQXCNVC - Convert characters" on page 613.

MQDCC_SOURCE_ENC_MASK	240	X'000000F0'
MQDCC_TARGET_ENC_MASK	3840	X'00000F00'
MQDCC_SOURCE_ENC_FACTOR	16	X'00000010'
MQDCC_TARGET_ENC_FACTOR	256	X'00000100'

MQDCC_* (Convert-characters options)

See the *Options* parameter described in "MQXCNVC - Convert characters" on page 613.

MQDCC_SOURCE_ENC_NATIVE	(variable)	
MQDCC_TARGET_ENC_NATIVE	(variable)	
MQDCC_NONE	0	X'00000000'
MQDCC_TARGET_ENC_UNDEFINED	0	X'00000000'
MQDCC_SOURCE_ENC_UNDEFINED	0	X'00000000'
MQDCC_DEFAULT_CONVERSION	1	X'00000001'
MQDCC_SOURCE_ENC_NORMAL	16	X'00000010'
MQDCC_FILL_TARGET_BUFFER	2	X'00000002'
MQDCC_TARGET_ENC_NORMAL	256	X'00000100'
MQDCC_SOURCE_ENC_REVERSED	32	X'00000020'
MQDCC_TARGET_ENC_REVERSED	512	X'00000200'

MQDH_* (Distribution header structure identifier)

See the *StrucId* field described in "Chapter 5. MQDH - Distribution header" on page 61.

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 5. MQDH - Distribution header" on page 61.

MQDH_VERSION_1	1	X'00000001'
MQDH_CURRENT_VERSION	1	X'00000001'

MQDHF_* (Distribution header flags)

See the *Flags* field described in "Chapter 5. MQDH - Distribution header" on page 61.

MQDHF_NONE	0	X'00000000'
MQDHF_NEW_MSG_IDS	1	X'00000001'

MQDL_* (Distribution list support)

See the *DistLists* attributes described in "Chapter 42. Attributes for the queue manager" on page 469 and "Chapter 39. Attributes for queues" on page 427.

MQDL_NOT_SUPPORTED	0	X'00000000'
MQDL_SUPPORTED	1	X'00000001'

MQDLH_* (Dead-letter header structure identifier)

See the *StrucId* field described in "Chapter 6. MQDLH - Dead-letter header" on page 69.

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 6. MQDLH - Dead-letter header" on page 69.

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 607.

MQDXP_STRUC_ID	'DXPb'

For the C programming language, the following array version is also defined:

MQDXP_STRUC_ID_ARRAY	'D','X','P','b'

MQDXP_* (Data-conversion-exit parameter structure version)

See the *Version* field described in "MQDXP – Data-conversion exit parameter" on page 607.

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 7. MQGMO - Get-message options" on page 81.

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 9. MQMD - Message descriptor" on page 125.

MQEI_UNLIMITED	-1	X'FFFFFFFF'

MQENC_* (Encoding)

See the *Encoding* field described in "Chapter 9. MQMD - Message descriptor" on page 125.

MQENC_NATIVE	(variable)

This constant has the following values in the environments indicated:

DOS client, Windows client	546
Windows 3.1, Windows 95, Windows 98	546
OS/2, Windows NT	546
Micro Focus COBOL on OS/2, Windows NT	17
Compaq (DIGITAL) OpenVMS	273
OS/390	785
AS/400	273
Tandem NonStop Kernel	273
UNIX systems (AIX, AT&T, HP-UX)	273

MQENC_* (Encoding masks)

See "Appendix D. Machine encodings" on page 589.

MQENC_INTEGER_MASK	15	X'0000000F'
MQENC_DECIMAL_MASK	240	X'000000F0'
MQENC_FLOAT_MASK	3840	X'00000F00'
MQENC_RESERVED_MASK	-4096	X'FFFFF000'

MQENC_* (Encoding for packed-decimal integers)

See “Appendix D. Machine encodings” on page 589.

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 589.

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 589.

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 39. Attributes for queues” on page 427, and the *AuthorityEvent*, *ChannelAutoDefEvent*, *InhibitEvent*, *LocalEvent*, *PerformanceEvent*, *RemoteEvent*, and *StartStopEvent* attributes described in “Chapter 42. Attributes for the queue manager” on page 469.

MQEVR_DISABLED	0	X'00000000'
MQEVR_ENABLED	1	X'00000001'

MQFB_* (Feedback)

See the *Feedback* field described in “Chapter 9. MQMD - Message descriptor” on page 125, and the *Reason* field described in “Chapter 6. MQDLH - Dead-letter header” on page 69; 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	999999999	X'3B9AC9FF'

MQFMT_* (Format)

See the *Format* field described in "Chapter 9. MQMD - Message descriptor" on page 125 and in other structures.

MQFMT_NONE	'bbbbbbbb'
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'
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 9. MQMD - Message descriptor" on page 125.

MQGI_NONE	X'00...00' (24 nulls)

For the C programming language, the following array version is also defined:

MQGI_NONE_ARRAY	'\0','\0',...'\0','\0'

MQGMO_* (Get message options)

See the *Options* field described in "Chapter 7. MQGMO - Get-message options" on page 81.

MQGMO_NONE	0	X'00000000'
MQGMO_NO_WAIT	0	X'00000000'
MQGMO_WAIT	1	X'00000001'
MQGMO_UNLOCK	1024	X'00000400'
MQGMO_MARK_SKIP_BACKOUT	128	X'00000080'
MQGMO_ALL_MSGS_AVAILABLE	131072	X'00020000'
MQGMO_BROWSE_FIRST	16	X'00000010'
MQGMO_CONVERT	16384	X'00004000'
MQGMO_SYNCPOINT	2	X'00000002'
MQGMO_BROWSE_MSG_UNDER_CURSOR	2048	X'00000800'
MQGMO_MSG_UNDER_CURSOR	256	X'00000100'
MQGMO_ALL_SEGMENTS_AVAILABLE	262144	X'00040000'
MQGMO_BROWSE_NEXT	32	X'00000020'

MQGMO_LOGICAL_ORDER	32768	X'00008000'
MQGMO_NO_SYNCPOINT	4	X'00000004'
MQGMO_SYNCPOINT_IF_PERSISTENT	4096	X'00001000'
MQGMO_LOCK	512	X'00000200'
MQGMO_ACCEPT_TRUNCATED_MSG	64	X'00000040'
MQGMO_COMPLETE_MSG	65536	X'00010000'
MQGMO_SET_SIGNAL	8	X'00000008'
MQGMO_FAIL_IF_QUIESCING	8192	X'00002000'

MQGMO_* (Get message options structure identifier)

See the *StrucId* field described in "Chapter 7. MQGMO - Get-message options" on page 81.

MQGMO_STRUC_ID	'GMOb'

For the C programming language, the following array version is also defined:

MQGMO_STRUC_ID_ARRAY	'G','M','O','b'

MQGMO_* (Get message options version)

See the *Version* field described in "Chapter 7. MQGMO - Get-message options" on page 81.

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 7. MQGMO - Get-message options" on page 81.

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 29. MQCONN - Connect queue manager" on page 333 and "Chapter 31. MQDISC - Disconnect queue manager" on page 345.

MQHC_UNUSABLE_HCONN	-1	X'FFFFFFFF'
MQHC_DEF_HCONN	0	X'00000000'

MQHO_* (Object handle)

See the *Hobj* parameter described in "Chapter 27. MQCLOSE - Close object" on page 319.

MQHO_UNUSABLE_HOBJ	-1	X'FFFFFFFF'
MQHO_NONE	0	X'00000000'

MQIA_* (Integer attribute selector)

See the *Selectors* parameter described in "Chapter 33. MQINQ - Inquire about object attributes" on page 363 and "Chapter 37. MQSET - Set object attributes" on page 415.

MQIA_LAST_USED	(variable)	
MQIA_FIRST	1	X'00000001'
MQIA_APPL_TYPE	1	X'00000001'
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'
MQIA_NAME_COUNT	19	X'00000013'
MQIA_CODED_CHAR_SET_ID	2	X'00000002'
MQIA_Q_TYPE	20	X'00000014'
MQIA_USER_LIST	2000	X'000007D0'
MQIA_LAST	2000	X'000007D0'
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_CURRENT_Q_DEPTH	3	X'00000003'
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_DEF_INPUT_OPEN_OPTION	4	X'00000004'
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_DEF_PERSISTENCE	5	X'00000005'
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_PRIORITY	6	X'00000006'
MQIA_ARCHIVE	60	X'0000003C'
MQIA_DEF_BIND	61	X'0000003D'
MQIA_QSG_DISP	63	X'0000003F'
MQIA_INTRA_GROUP_QUEUING	64	X'00000040'
MQIA_IGQ_PUT_AUTHORITY	65	X'00000041'
MQIA_DEFINITION_TYPE	7	X'00000007'
MQIA_HARDEN_GET_BACKOUT	8	X'00000008'
MQIA_INHIBIT_GET	9	X'00000009'

MQIAUT_* (IMS authenticator)

See the *Authenticator* field described in "Chapter 8. MQIIH - IMS information header" on page 117.

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 33. MQINQ - Inquire about object attributes" on page 363.

MQIAV_NOT_APPLICABLE	-1	X'FFFFFFFF'
MQIAV_UNDEFINED	-2	X'FFFFFFFE'

MQICM_* (IMS commit mode)

See the *CommitMode* field described in "Chapter 8. MQIIH - IMS information header" on page 117.

MQICM_COMMIT_THEN_SEND	'0'
MQICM_SEND_THEN_COMMIT	'1'

MQIGQ_* (Intra-group queuing)

See the *IntraGroupQueuing* attribute described in "Chapter 42. Attributes for the queue manager" on page 469.

MQIGQ_DISABLED	0	X'00000000'
MQIGQ_ENABLED	1	X'00000001'

MQIGQPA_* (Intra-group queuing put authority)

See the *IGQPutAuthority* attribute described in "Chapter 42. Attributes for the queue manager" on page 469.

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 8. MQIIH - IMS information header" on page 117.

MQIIH_NONE	0	X'00000000'

MQIIH_* (IMS header length)

See the *StrucLength* field described in "Chapter 8. MQIIH - IMS information header" on page 117.

MQIIH_LENGTH_1	84	X'00000054'

MQIIH_* (IMS header structure identifier)

See the *StrucId* field described in "Chapter 8. MQIIH - IMS information header" on page 117.

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 8. MQIIH - IMS information header" on page 117.

MQIIH_VERSION_1	1	X'00000001'
MQIIH_CURRENT_VERSION	1	X'00000001'

MQISS_* (IMS security scope)

See the *SecurityScope* field described in "Chapter 8. MQIIH - IMS information header" on page 117.

MQISS_CHECK	'C'
MQISS_FULL	'F'

MQIT_* (Index type)

See the *IndexType* attribute described in "Chapter 39. Attributes for queues" on page 427.

MQIT_NONE	0	X'00000000'
MQIT_MSG_ID	1	X'00000001'

MQIT_CORREL_ID	2	X'00000002'
MQIT_MSG_TOKEN	4	X'00000004'

MQITII_* (IMS transaction instance identifier)

See the *TranInstanceId* field described in "Chapter 8. MQIIH - IMS information header" on page 117.

MQITII_NONE	X'00...00' (16 nulls)

For the C programming language, the following array version is also defined:

MQITII_NONE_ARRAY	'\0','\0',...'\0','\0'

MQITS_* (IMS transaction state)

See the *TranState* field described in "Chapter 8. MQIIH - IMS information header" on page 117.

MQITS_IN_CONVERSATION	'C'
MQITS_NOT_IN_CONVERSATION	' '
MQITS_ARCHITECTED	'A'

MQMD_* (Message descriptor structure identifier)

See the *StrucId* field described in "Chapter 9. MQMD - Message descriptor" on page 125.

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 9. MQMD - Message descriptor" on page 125.

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 10. MQMDE - Message descriptor extension" on page 185.

MQMDE_LENGTH_2	72	X'00000048'

MQMDE_* (Message descriptor extension structure identifier)

See the *StrucId* field described in "Chapter 10. MQMDE - Message descriptor extension" on page 185.

MQMDE_STRUC_ID	'MDEb'

For the C programming language, the following array version is also defined:

MQMDE_STRUC_ID_ARRAY	'M','D','E','b'

MQMDE_* (Message descriptor extension version)

See the *Version* field described in "Chapter 10. MQMDE - Message descriptor extension" on page 185.

MQMDE_VERSION_2	2	X'00000002'
MQMDE_CURRENT_VERSION	2	X'00000002'

MQMDEF_* (Message descriptor extension flags)

See the *Flags* field described in "Chapter 10. MQMDE - Message descriptor extension" on page 185.

MQMDEF_NONE	0	X'00000000'

MQMDS_* (Message delivery sequence)

See the *MsgDeliverySequence* attribute described in "Chapter 39. Attributes for queues" on page 427.

MQMDS_PRIORITY	0	X'00000000'
MQMDS_FIFO	1	X'00000001'

MQMF_* (Message flags)

See the *MsgFlags* field described in "Chapter 9. MQMD - Message descriptor" on page 125.

MQMF_NONE	0	X'00000000'
MQMF_SEGMENTATION_INHIBITED	0	X'00000000'
MQMF_SEGMENTATION_ALLOWED	1	X'00000001'
MQMF_LAST_MSG_IN_GROUP	16	X'00000010'
MQMF_SEGMENT	2	X'00000002'
MQMF_LAST_SEGMENT	4	X'00000004'
MQMF_MSG_IN_GROUP	8	X'00000008'

MQMF_* (Message-flags masks)

See "Appendix E. Report options and message flags" on page 593.

MQMF_ACCEPT_UNSUP_MASK	-1048576	X'FFF00000'
MQMF_ACCEPT_UNSUP_IF_XMIT_MASK	1044480	X'000FF000'
MQMF_REJECT_UNSUP_MASK	4095	X'00000FFF'

MQMI_* (Message identifier)

See the *MsgId* field described in "Chapter 9. MQMD - Message descriptor" on page 125.

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 7. MQGMO - Get-message options" on page 81.

MQMO_NONE	0	X'00000000'
MQMO_MATCH_MSG_ID	1	X'00000001'
MQMO_MATCH_OFFSET	16	X'00000010'
MQMO_MATCH_CORREL_ID	2	X'00000002'
MQMO_MATCH_MSG_TOKEN	32	X'00000020'
MQMO_MATCH_GROUP_ID	4	X'00000004'
MQMO_MATCH_MSG_SEQ_NUMBER	8	X'00000008'

MQMT_* (Message type)

See the *MsgType* field described in "Chapter 9. MQMD - Message descriptor" on page 125.

MQMT_SYSTEM_FIRST	1	X'00000001'
MQMT_REQUEST	1	X'00000001'
MQMT_MQE_FIELDS_FROM_MQE	112	X'00000070'
MQMT_MQE_FIELDS	113	X'00000071'
MQMT_REPLY	2	X'00000002'
MQMT_REPORT	4	X'00000004'
MQMT_SYSTEM_LAST	65535	X'0000FFFF'
MQMT_APPL_FIRST	65536	X'00010000'
MQMT_DATAGRAM	8	X'00000008'
MQMT_APPL_LAST	999999999	X'3B9AC9FF'

MQMTOK_* (Message token)

See the *MsgToken* fields described in "Chapter 7. MQGMO - Get-message options" on page 81 and "Chapter 21. MQWIH - Work information header" on page 279.

MQMTOK_NONE	X'00...00' (16 nulls)

For the C programming language, the following array version is also defined:

MQMTOK_NONE_ARRAY	'\0','\0',...'\0','\0'

MQNC_* (Name count)

See the *NameCount* attribute described in "Chapter 40. Attributes for namelists" on page 459.

MQNC_MAX_NAMELIST_NAME_COUNT	256	X'00000100'

MQOD_* (Object descriptor length)

See "Chapter 11. MQOD - Object descriptor" on page 193.

MQOD_CURRENT_LENGTH	(variable)

MQOD_* (Object descriptor structure identifier)

See the *StrucId* field described in "Chapter 11. MQOD - Object descriptor" on page 193.

MQOD_STRUC_ID	'ODbb'

For the C programming language, the following array version is also defined:

MQOD_STRUC_ID_ARRAY	'O','D','b','b'

MQOD_* (Object descriptor version)

See the *Version* field described in "Chapter 11. MQOD - Object descriptor" on page 193.

MQOD_VERSION_1	1	X'00000001'
MQOD_VERSION_2	2	X'00000002'
MQOD_VERSION_3	3	X'00000003'
MQOD_CURRENT_VERSION	(variable)	

MQOII_* (Object instance identifier)

See the *ObjectInstanceId* field described in "Chapter 17. MQRMH - Reference message header" on page 251.

MQOII_NONE	X'00...00' (24 nulls)

For the C programming language, the following array version is also defined:

MQOII_NONE_ARRAY	'\0','\0',...'\0','\0'

MQOL_* (Original length)

See the *OriginalLength* field described in "Chapter 9. MQMD - Message descriptor" on page 125.

MQOL_UNDEFINED	-1	X'FFFFFFFF'

MQOO_* (Open options)

See the *Options* parameter described in "Chapter 34. MQOPEN - Open object" on page 375.

MQOO_BIND_AS_Q_DEF	0	X'00000000'
MQOO_INPUT_AS_Q_DEF	1	X'00000001'
MQOO_SET_IDENTITY_CONTEXT	1024	X'00000400'
MQOO_SAVE_ALL_CONTEXT	128	X'00000080'
MQOO_OUTPUT	16	X'00000010'
MQOO_BIND_ON_OPEN	16384	X'00004000'
MQOO_INPUT_SHARED	2	X'00000002'
MQOO_SET_ALL_CONTEXT	2048	X'00000800'
MQOO_PASS_IDENTITY_CONTEXT	256	X'00000100'
MQOO_INQUIRE	32	X'00000020'
MQOO_BIND_NOT_FIXED	32768	X'00008000'
MQOO_INPUT_EXCLUSIVE	4	X'00000004'
MQOO_ALTERNATE_USER_AUTHORITY	4096	X'00001000'
MQOO_PASS_ALL_CONTEXT	512	X'00000200'
MQOO_SET	64	X'00000040'
MQOO_RESOLVE_NAMES	65536	X'00010000'
MQOO_BROWSE	8	X'00000008'
MQOO_FAIL_IF_QUIESCING	8192	X'00002000'

MQOT_* (Object type)

See the *ObjectType* field described in "Chapter 11. MQOD - Object descriptor" on page 193.

MQOT_Q	1	X'00000001'
MQOT_NAMELIST	2	X'00000002'
MQOT_PROCESS	3	X'00000003'
MQOT_Q_MGR	5	X'00000005'
MQOT_CHANNEL	6	X'00000006'
MQOT_RESERVED_1	7	X'00000007'

MQPER_* (Persistence)

See the *Persistence* field described in "Chapter 9. MQMD - Message descriptor" on page 125, and the *DefPersistence* attribute described in "Chapter 39. Attributes for queues" on page 427.

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 42. Attributes for the queue manager" on page 469.

MQPL_MVS	1	X'00000001'
MQPL_OS390	1	X'00000001'
MQPL_WINDOWS_NT	11	X'0000000B'
MQPL_VMS	12	X'0000000C'
MQPL_NSK	13	X'0000000D'
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'

MQPMO_* (Put message options)

See the *Options* field described in "Chapter 13. MQPMO - Put message options" on page 211.

MQPMO_NONE	0	X'00000000'
MQPMO_SET_IDENTITY_CONTEXT	1024	X'00000400'
MQPMO_NEW_CORREL_ID	128	X'00000080'
MQPMO_NO_CONTEXT	16384	X'00004000'
MQPMO_SYNCPOINT	2	X'00000002'
MQPMO_SET_ALL_CONTEXT	2048	X'00000800'
MQPMO_PASS_IDENTITY_CONTEXT	256	X'00000100'
MQPMO_DEFAULT_CONTEXT	32	X'00000020'
MQPMO_LOGICAL_ORDER	32768	X'00008000'
MQPMO_NO_SYNCPOINT	4	X'00000004'
MQPMO_ALTERNATE_USER_AUTHORITY	4096	X'00001000'
MQPMO_PASS_ALL_CONTEXT	512	X'00000200'
MQPMO_NEW_MSG_ID	64	X'00000040'
MQPMO_FAIL_IF_QUIESCING	8192	X'00002000'

MQPMO_* (Put message options structure length)

See "Chapter 13. MQPMO - Put message options" on page 211.

MQPMO_CURRENT_LENGTH	(variable)

MQPMO_* (Put message options structure identifier)

See the *StrucId* field described in "Chapter 13. MQPMO - Put message options" on page 211.

MQPMO_STRUC_ID	'PMOb'

For the C programming language, the following array version is also defined:

MQPMO_STRUC_ID_ARRAY	'P','M','O','b'

MQPMO_* (Put message options version)

See the *Version* field described in "Chapter 13. MQPMO - Put message options" on page 211.

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 5. MQDH - Distribution header" on page 61.

MQPMRF_NONE	0	X'00000000'
MQPMRF_MSG_ID	1	X'00000001'
MQPMRF_ACCOUNTING_TOKEN	16	X'00000010'
MQPMRF_CORREL_ID	2	X'00000002'
MQPMRF_GROUP_ID	4	X'00000004'
MQPMRF_FEEDBACK	8	X'00000008'

MQPRI_* (Priority)

See the *Priority* field described in "Chapter 9. MQMD - Message descriptor" on page 125.

MQPRI_PRIORITY_AS_Q_DEF	-1	X'FFFFFFFF'

MQQA_* (Inhibit get)

See the *InhibitGet* attribute described in "Chapter 39. Attributes for queues" on page 427.

MQQA_GET_ALLOWED	0	X'00000000'
MQQA_GET_INHIBITED	1	X'00000001'

MQQA_* (Inhibit put)

See the *InhibitPut* attribute described in "Chapter 39. Attributes for queues" on page 427.

MQQA_PUT_ALLOWED	0	X'00000000'
MQQA_PUT_INHIBITED	1	X'00000001'

MQQA_* (Backout hardening)

See the *HardenGetBackout* attribute described in "Chapter 39. Attributes for queues" on page 427.

MQQA_BACKOUT_NOT_HARDENED	0	X'00000000'
MQQA_BACKOUT_HARDENED	1	X'00000001'

MQQA_* (Queue shareability)

See the *Shareability* attribute described in "Chapter 39. Attributes for queues" on page 427.

MQQA_NOT_SHAREABLE	0	X'00000000'
MQQA_SHAREABLE	1	X'00000001'

MQQDT_* (Queue definition type)

See the *DefinitionType* attribute described in "Chapter 39. Attributes for queues" on page 427.

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 39. Attributes for queues" on page 427.

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 39. Attributes for queues" on page 427.

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 39. Attributes for queues" on page 427.

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 489, and the *Feedback* field described in "Chapter 9. MQMD - Message descriptor" on page 125. Note: the following list is in **numeric order**.

MQRC_NONE	0	X'00000000'
MQRC_ALIAS_BASE_Q_TYPE_ERROR	2001	X'000007D1'
MQRC_ALREADY_CONNECTED	2002	X'000007D2'
MQRC_BACKED_OUT	2003	X'000007D3'
MQRC_BUFFER_ERROR	2004	X'000007D4'
MQRC_BUFFER_LENGTH_ERROR	2005	X'000007D5'
MQRC_CHAR_ATTR_LENGTH_ERROR	2006	X'000007D6'
MQRC_CHAR_ATTRS_ERROR	2007	X'000007D7'
MQRC_CHAR_ATTRS_TOO_SHORT	2008	X'000007D8'
MQRC_CONNECTION_BROKEN	2009	X'000007D9'
MQRC_DATA_LENGTH_ERROR	2010	X'000007DA'
MQRC_DYNAMIC_Q_NAME_ERROR	2011	X'000007DB'
MQRC_ENVIRONMENT_ERROR	2012	X'000007DC'
MQRC_EXPIRY_ERROR	2013	X'000007DD'
MQRC_FEEDBACK_ERROR	2014	X'000007DE'
MQRC_GET_INHIBITED	2016	X'000007E0'
MQRC_HANDLE_NOT_AVAILABLE	2017	X'000007E1'
MQRC_HCONN_ERROR	2018	X'000007E2'
MQRC_HOBJ_ERROR	2019	X'000007E3'
MQRC_INHIBIT_VALUE_ERROR	2020	X'000007E4'
MQRC_INT_ATTR_COUNT_ERROR	2021	X'000007E5'
MQRC_INT_ATTR_COUNT_TOO_SMALL	2022	X'000007E6'
MQRC_INT_ATTRS_ARRAY_ERROR	2023	X'000007E7'

MQRC_SYNCPOINT_LIMIT_REACHED	2024	X'000007E8'
MQRC_MAX_CONNS_LIMIT_REACHED	2025	X'000007E9'
MQRC_MD_ERROR	2026	X'000007EA'
MQRC_MISSING_REPLY_TO_Q	2027	X'000007EB'
MQRC_MSG_TYPE_ERROR	2029	X'000007ED'
MQRC_MSG_TOO_BIG_FOR_Q	2030	X'000007EE'
MQRC_MSG_TOO_BIG_FOR_Q_MGR	2031	X'000007EF'
MQRC_NO_MSG_AVAILABLE	2033	X'000007F1'
MQRC_NO_MSG_UNDER_CURSOR	2034	X'000007F2'
MQRC_NOT_AUTHORIZED	2035	X'000007F3'
MQRC_NOT_OPEN_FOR_BROWSE	2036	X'000007F4'
MQRC_NOT_OPEN_FOR_INPUT	2037	X'000007F5'
MQRC_NOT_OPEN_FOR_INQUIRE	2038	X'000007F6'
MQRC_NOT_OPEN_FOR_OUTPUT	2039	X'000007F7'
MQRC_NOT_OPEN_FOR_SET	2040	X'000007F8'
MQRC_OBJECT_CHANGED	2041	X'000007F9'
MQRC_OBJECT_IN_USE	2042	X'000007FA'
MQRC_OBJECT_TYPE_ERROR	2043	X'000007FB'
MQRC_OD_ERROR	2044	X'000007FC'
MQRC_OPTION_NOT_VALID_FOR_TYPE	2045	X'000007FD'
MQRC_OPTIONS_ERROR	2046	X'000007FE'
MQRC_PERSISTENCE_ERROR	2047	X'000007FF'
MQRC_PERSISTENT_NOT_ALLOWED	2048	X'00000800'
MQRC_PRIORITY_EXCEEDS_MAXIMUM	2049	X'00000801'
MQRC_PRIORITY_ERROR	2050	X'00000802'
MQRC_PUT_INHIBITED	2051	X'00000803'
MQRC_Q_DELETED	2052	X'00000804'
MQRC_Q_FULL	2053	X'00000805'
MQRC_Q_NOT_EMPTY	2055	X'00000807'
MQRC_Q_SPACE_NOT_AVAILABLE	2056	X'00000808'
MQRC_Q_TYPE_ERROR	2057	X'00000809'
MQRC_Q_MGR_NAME_ERROR	2058	X'0000080A'
MQRC_Q_MGR_NOT_AVAILABLE	2059	X'0000080B'
MQRC_REPORT_OPTIONS_ERROR	2061	X'0000080D'
MQRC_SECOND_MARK_NOT_ALLOWED	2062	X'0000080E'
MQRC_SECURITY_ERROR	2063	X'0000080F'
MQRC_SELECTOR_COUNT_ERROR	2065	X'00000811'
MQRC_SELECTOR_LIMIT_EXCEEDED	2066	X'00000812'
MQRC_SELECTOR_ERROR	2067	X'00000813'
MQRC_SELECTOR_NOT_FOR_TYPE	2068	X'00000814'
MQRC_SIGNAL_OUTSTANDING	2069	X'00000815'
MQRC_SIGNAL_REQUEST_ACCEPTED	2070	X'00000816'
MQRC_STORAGE_NOT_AVAILABLE	2071	X'00000817'
MQRC_SYNCPOINT_NOT_AVAILABLE	2072	X'00000818'
MQRC_TRIGGER_CONTROL_ERROR	2075	X'0000081B'
MQRC_TRIGGER_DEPTH_ERROR	2076	X'0000081C'
MQRC_TRIGGER_MSG_PRIORITY_ERR	2077	X'0000081D'
MQRC_TRIGGER_TYPE_ERROR	2078	X'0000081E'
MQRC_TRUNCATED_MSG_ACCEPTED	2079	X'0000081F'
MQRC_TRUNCATED_MSG_FAILED	2080	X'00000820'
MQRC_UNKNOWN_ALIAS_BASE_Q	2082	X'00000822'
MQRC_UNKNOWN_OBJECT_NAME	2085	X'00000825'
MQRC_UNKNOWN_OBJECT_Q_MGR	2086	X'00000826'
MQRC_UNKNOWN_REMOTE_Q_MGR	2087	X'00000827'
MQRC_WAIT_INTERVAL_ERROR	2090	X'0000082A'
MQRC_XMIT_Q_TYPE_ERROR	2091	X'0000082B'
MQRC_XMIT_Q_USAGE_ERROR	2092	X'0000082C'
MQRC_NOT_OPEN_FOR_PASS_ALL	2093	X'0000082D'
MQRC_NOT_OPEN_FOR_PASS_IDENT	2094	X'0000082E'

MQRC_NOT_OPEN_FOR_SET_ALL	2095	X'0000082F'
MQRC_NOT_OPEN_FOR_SET_IDENT	2096	X'00000830'
MQRC_CONTEXT_HANDLE_ERROR	2097	X'00000831'
MQRC_CONTEXT_NOT_AVAILABLE	2098	X'00000832'
MQRC_SIGNAL1_ERROR	2099	X'00000833'
MQRC_OBJECT_ALREADY_EXISTS	2100	X'00000834'
MQRC_OBJECT_DAMAGED	2101	X'00000835'
MQRC_RESOURCE_PROBLEM	2102	X'00000836'
MQRC_ANOTHER_Q_MGR_CONNECTED	2103	X'00000837'
MQRC_UNKNOWN_REPORT_OPTION	2104	X'00000838'
MQRC_STORAGE_CLASS_ERROR	2105	X'00000839'
MQRC_COD_NOT_VALID_FOR_XCF_Q	2106	X'0000083A'
MQRC_XWAIT_CANCELED	2107	X'0000083B'
MQRC_XWAIT_ERROR	2108	X'0000083C'
MQRC_SUPPRESSED_BY_EXIT	2109	X'0000083D'
MQRC_FORMAT_ERROR	2110	X'0000083E'
MQRC_SOURCE_CCSID_ERROR	2111	X'0000083F'
MQRC_SOURCE_INTEGER_ENC_ERROR	2112	X'00000840'
MQRC_SOURCE_DECIMAL_ENC_ERROR	2113	X'00000841'
MQRC_SOURCE_FLOAT_ENC_ERROR	2114	X'00000842'
MQRC_TARGET_CCSID_ERROR	2115	X'00000843'
MQRC_TARGET_INTEGER_ENC_ERROR	2116	X'00000844'
MQRC_TARGET_DECIMAL_ENC_ERROR	2117	X'00000845'
MQRC_TARGET_FLOAT_ENC_ERROR	2118	X'00000846'
MQRC_NOT_CONVERTED	2119	X'00000847'
MQRC_CONVERTED_MSG_TOO_BIG	2120	X'00000848'
MQRC_TRUNCATED	2120	X'00000848'
MQRC_NO_EXTERNAL_PARTICIPANTS	2121	X'00000849'
MQRC_PARTICIPANT_NOT_AVAILABLE	2122	X'0000084A'
MQRC_OUTCOME_MIXED	2123	X'0000084B'
MQRC_OUTCOME_PENDING	2124	X'0000084C'
MQRC_BRIDGE_STARTED	2125	X'0000084D'
MQRC_BRIDGE_STOPPED	2126	X'0000084E'
MQRC_ADAPTER_STORAGE_SHORTAGE	2127	X'0000084F'
MQRC_UOW_IN_PROGRESS	2128	X'00000850'
MQRC_ADAPTER_CONN_LOAD_ERROR	2129	X'00000851'
MQRC_ADAPTER_SERV_LOAD_ERROR	2130	X'00000852'
MQRC_ADAPTER_DEFS_ERROR	2131	X'00000853'
MQRC_ADAPTER_DEFS_LOAD_ERROR	2132	X'00000854'
MQRC_ADAPTER_CONV_LOAD_ERROR	2133	X'00000855'
MQRC_BO_ERROR	2134	X'00000856'
MQRC_DH_ERROR	2135	X'00000857'
MQRC_MULTIPLE_REASONS	2136	X'00000858'
MQRC_OPEN_FAILED	2137	X'00000859'
MQRC_ADAPTER_DISC_LOAD_ERROR	2138	X'0000085A'
MQRC_CNO_ERROR	2139	X'0000085B'
MQRC_CICS_WAIT_FAILED	2140	X'0000085C'
MQRC_DLH_ERROR	2141	X'0000085D'
MQRC_HEADER_ERROR	2142	X'0000085E'
MQRC_SOURCE_LENGTH_ERROR	2143	X'0000085F'
MQRC_TARGET_LENGTH_ERROR	2144	X'00000860'
MQRC_SOURCE_BUFFER_ERROR	2145	X'00000861'
MQRC_TARGET_BUFFER_ERROR	2146	X'00000862'
MQRC_IIH_ERROR	2148	X'00000864'
MQRC_PCF_ERROR	2149	X'00000865'
MQRC_DBCS_ERROR	2150	X'00000866'
MQRC_OBJECT_NAME_ERROR	2152	X'00000868'
MQRC_OBJECT_Q_MGR_NAME_ERROR	2153	X'00000869'
MQRC_RECS_PRESENT_ERROR	2154	X'0000086A'

MQRC_OBJECT_RECORDS_ERROR	2155	X'0000086B'
MQRC_RESPONSE_RECORDS_ERROR	2156	X'0000086C'
MQRC_ASID_MISMATCH	2157	X'0000086D'
MQRC_PMO_RECORD_FLAGS_ERROR	2158	X'0000086E'
MQRC_PUT_MSG_RECORDS_ERROR	2159	X'0000086F'
MQRC_CONN_ID_IN_USE	2160	X'00000870'
MQRC_Q_MGR_QUIESCING	2161	X'00000871'
MQRC_Q_MGR_STOPPING	2162	X'00000872'
MQRC_DUPLICATE_RECOV_COORD	2163	X'00000873'
MQRC_PMO_ERROR	2173	X'0000087D'
MQRC_API_EXIT_LOAD_ERROR	2183	X'00000887'
MQRC_REMOTE_Q_NAME_ERROR	2184	X'00000888'
MQRC_INCONSISTENT_PERSISTENCE	2185	X'00000889'
MQRC_GMO_ERROR	2186	X'0000088A'
MQRC_CICS_BRIDGE_RESTRICTION	2187	X'0000088B'
MQRC_STOPPED_BY_CLUSTER_EXIT	2188	X'0000088C'
MQRC_CLUSTER_RESOLUTION_ERROR	2189	X'0000088D'
MQRC_CONVERTED_STRING_TOO_BIG	2190	X'0000088E'
MQRC_TMC_ERROR	2191	X'0000088F'
MQRC_PAGESET_FULL	2192	X'00000890'
MQRC_STORAGE_MEDIUM_FULL	2192	X'00000890'
MQRC_PAGESET_ERROR	2193	X'00000891'
MQRC_NAME_NOT_VALID_FOR_TYPE	2194	X'00000892'
MQRC_UNEXPECTED_ERROR	2195	X'00000893'
MQRC_UNKNOWN_XMIT_Q	2196	X'00000894'
MQRC_UNKNOWN_DEF_XMIT_Q	2197	X'00000895'
MQRC_DEF_XMIT_Q_TYPE_ERROR	2198	X'00000896'
MQRC_DEF_XMIT_Q_USAGE_ERROR	2199	X'00000897'
MQRC_NAME_IN_USE	2201	X'00000899'
MQRC_CONNECTION_QUIESCING	2202	X'0000089A'
MQRC_CONNECTION_STOPPING	2203	X'0000089B'
MQRC_ADAPTER_NOT_AVAILABLE	2204	X'0000089C'
MQRC_MSG_ID_ERROR	2206	X'0000089E'
MQRC_CORREL_ID_ERROR	2207	X'0000089F'
MQRC_FILE_SYSTEM_ERROR	2208	X'000008A0'
MQRC_NO_MSG_LOCKED	2209	X'000008A1'
MQRC_FILE_NOT_AUDITED	2216	X'000008A8'
MQRC_CONNECTION_NOT_AUTHORIZED	2217	X'000008A9'
MQRC_MSG_TOO_BIG_FOR_CHANNEL	2218	X'000008AA'
MQRC_CALL_IN_PROGRESS	2219	X'000008AB'
MQRC_RMH_ERROR	2220	X'000008AC'
MQRC_Q_MGR_ACTIVE	2222	X'000008AE'
MQRC_Q_MGR_NOT_ACTIVE	2223	X'000008AF'
MQRC_Q_DEPTH_HIGH	2224	X'000008B0'
MQRC_Q_DEPTH_LOW	2225	X'000008B1'
MQRC_Q_SERVICE_INTERVAL_HIGH	2226	X'000008B2'
MQRC_Q_SERVICE_INTERVAL_OK	2227	X'000008B3'
MQRC_UNIT_OF_WORK_NOT_STARTED	2232	X'000008B8'
MQRC_CHANNEL_AUTO_DEF_OK	2233	X'000008B9'
MQRC_CHANNEL_AUTO_DEF_ERROR	2234	X'000008BA'
MQRC_CFH_ERROR	2235	X'000008BB'
MQRC_CFIL_ERROR	2236	X'000008BC'
MQRC_CFIN_ERROR	2237	X'000008BD'
MQRC_CFSL_ERROR	2238	X'000008BE'
MQRC_CFST_ERROR	2239	X'000008BF'
MQRC_INCOMPLETE_GROUP	2241	X'000008C1'
MQRC_INCOMPLETE_MSG	2242	X'000008C2'
MQRC_INCONSISTENT_CCSIDS	2243	X'000008C3'
MQRC_INCONSISTENT_ENCODINGS	2244	X'000008C4'

MQRC_INCONSISTENT_UOW	2245	X'000008C5'
MQRC_INVALID_MSG_UNDER_CURSOR	2246	X'000008C6'
MQRC_MATCH_OPTIONS_ERROR	2247	X'000008C7'
MQRC_MDE_ERROR	2248	X'000008C8'
MQRC_MSG_FLAGS_ERROR	2249	X'000008C9'
MQRC_MSG_SEQ_NUMBER_ERROR	2250	X'000008CA'
MQRC_OFFSET_ERROR	2251	X'000008CB'
MQRC_ORIGINAL_LENGTH_ERROR	2252	X'000008CC'
MQRC_SEGMENT_LENGTH_ZERO	2253	X'000008CD'
MQRC_UOW_NOT_AVAILABLE	2255	X'000008CF'
MQRC_WRONG_GMO_VERSION	2256	X'000008D0'
MQRC_WRONG_MD_VERSION	2257	X'000008D1'
MQRC_GROUP_ID_ERROR	2258	X'000008D2'
MQRC_INCONSISTENT_BROWSE	2259	X'000008D3'
MQRC_XQH_ERROR	2260	X'000008D4'
MQRC_SRC_ENV_ERROR	2261	X'000008D5'
MQRC_SRC_NAME_ERROR	2262	X'000008D6'
MQRC_DEST_ENV_ERROR	2263	X'000008D7'
MQRC_DEST_NAME_ERROR	2264	X'000008D8'
MQRC_TM_ERROR	2265	X'000008D9'
MQRC_CLUSTER_EXIT_ERROR	2266	X'000008DA'
MQRC_CLUSTER_EXIT_LOAD_ERROR	2267	X'000008DB'
MQRC_CLUSTER_PUT_INHIBITED	2268	X'000008DC'
MQRC_CLUSTER_RESOURCE_ERROR	2269	X'000008DD'
MQRC_NO_DESTINATIONS_AVAILABLE	2270	X'000008DE'
MQRC_CONN_TAG_IN_USE	2271	X'000008DF'
MQRC_CONNECTION_ERROR	2273	X'000008E1'
MQRC_OPTION_ENVIRONMENT_ERROR	2274	X'000008E2'
MQRC_CD_ERROR	2277	X'000008E5'
MQRC_CLIENT_CONN_ERROR	2278	X'000008E6'
MQRC_CHANNEL_STOPPED_BY_USER	2279	X'000008E7'
MQRC_HCONFIG_ERROR	2280	X'000008E8'
MQRC_FUNCTION_ERROR	2281	X'000008E9'
MQRC_CHANNEL_STARTED	2282	X'000008EA'
MQRC_CHANNEL_STOPPED	2283	X'000008EB'
MQRC_CHANNEL_CONV_ERROR	2284	X'000008EC'
MQRC_SERVICE_NOT_AVAILABLE	2285	X'000008ED'
MQRC_INITIALIZATION_FAILED	2286	X'000008EE'
MQRC_TERMINATION_FAILED	2287	X'000008EF'
MQRC_UNKNOWN_Q_NAME	2288	X'000008F0'
MQRC_SERVICE_ERROR	2289	X'000008F1'
MQRC_Q_ALREADY_EXISTS	2290	X'000008F2'
MQRC_USER_ID_NOT_AVAILABLE	2291	X'000008F3'
MQRC_UNKNOWN_ENTITY	2292	X'000008F4'
MQRC_UNKNOWN_AUTH_ENTITY	2293	X'000008F5'
MQRC_UNKNOWN_REF_OBJECT	2294	X'000008F6'
MQRC_CHANNEL_ACTIVATED	2295	X'000008F7'
MQRC_CHANNEL_NOT_ACTIVATED	2296	X'000008F8'
MQRC_UOW_CANCELED	2297	X'000008F9'
MQRC_FUNCTION_NOT_SUPPORTED	2298	X'000008FA'
MQRC_SELECTOR_TYPE_ERROR	2299	X'000008FB'
MQRC_COMMAND_TYPE_ERROR	2300	X'000008FC'
MQRC_MULTIPLE_INSTANCE_ERROR	2301	X'000008FD'
MQRC_SYSTEM_ITEM_NOT_ALTERABLE	2302	X'000008FE'
MQRC_BAG_CONVERSION_ERROR	2303	X'000008FF'
MQRC_SELECTOR_OUT_OF_RANGE	2304	X'00000900'
MQRC_SELECTOR_NOT_UNIQUE	2305	X'00000901'
MQRC_INDEX_NOT_PRESENT	2306	X'00000902'
MQRC_STRING_ERROR	2307	X'00000903'

MQRC_ENCODING_NOT_SUPPORTED	2308	X'00000904'
MQRC_SELECTOR_NOT_PRESENT	2309	X'00000905'
MQRC_OUT_SELECTOR_ERROR	2310	X'00000906'
MQRC_STRING_TRUNCATED	2311	X'00000907'
MQRC_SELECTOR_WRONG_TYPE	2312	X'00000908'
MQRC_INCONSISTENT_ITEM_TYPE	2313	X'00000909'
MQRC_INDEX_ERROR	2314	X'0000090A'
MQRC_SYSTEM_BAG_NOT_ALTERABLE	2315	X'0000090B'
MQRC_ITEM_COUNT_ERROR	2316	X'0000090C'
MQRC_FORMAT_NOT_SUPPORTED	2317	X'0000090D'
MQRC_SELECTOR_NOT_SUPPORTED	2318	X'0000090E'
MQRC_ITEM_VALUE_ERROR	2319	X'0000090F'
MQRC_HBAG_ERROR	2320	X'00000910'
MQRC_PARAMETER_MISSING	2321	X'00000911'
MQRC_CMD_SERVER_NOT_AVAILABLE	2322	X'00000912'
MQRC_STRING_LENGTH_ERROR	2323	X'00000913'
MQRC_INQUIRY_COMMAND_ERROR	2324	X'00000914'
MQRC_NESTED_BAG_NOT_SUPPORTED	2325	X'00000915'
MQRC_BAG_WRONG_TYPE	2326	X'00000916'
MQRC_ITEM_TYPE_ERROR	2327	X'00000917'
MQRC_SYSTEM_BAG_NOT_DELETABLE	2328	X'00000918'
MQRC_SYSTEM_ITEM_NOT_DELETABLE	2329	X'00000919'
MQRC_CODED_CHAR_SET_ID_ERROR	2330	X'0000091A'
MQRC_MSG_TOKEN_ERROR	2331	X'0000091B'
MQRC_MISSING_WIH	2332	X'0000091C'
MQRC_WIH_ERROR	2333	X'0000091D'
MQRC_RFH_ERROR	2334	X'0000091E'
MQRC_RFH_STRING_ERROR	2335	X'0000091F'
MQRC_RFH_COMMAND_ERROR	2336	X'00000920'
MQRC_RFH_PARM_ERROR	2337	X'00000921'
MQRC_RFH_DUPLICATE_PARM	2338	X'00000922'
MQRC_RFH_PARM_MISSING	2339	X'00000923'
MQRC_CHAR_CONVERSION_ERROR	2340	X'00000924'
MQRC_UCS2_CONVERSION_ERROR	2341	X'00000925'
MQRC_DB2_NOT_AVAILABLE	2342	X'00000926'
MQRC_OBJECT_NOT_UNIQUE	2343	X'00000927'
MQRC_CONN_TAG_NOT_RELEASED	2344	X'00000928'
MQRC_CF_NOT_AVAILABLE	2345	X'00000929'
MQRC_CF_STRUC_IN_USE	2346	X'0000092A'
MQRC_CF_STRUC_LIST_HDR_IN_USE	2347	X'0000092B'
MQRC_CF_STRUC_AUTH_FAILED	2348	X'0000092C'
MQRC_CF_STRUC_ERROR	2349	X'0000092D'
MQRC_CONN_TAG_NOT_USABLE	2350	X'0000092E'
MQRC_GLOBAL_UOW_CONFLICT	2351	X'0000092F'
MQRC_LOCAL_UOW_CONFLICT	2352	X'00000930'
MQRC_HANDLE_IN_USE_FOR_UOW	2353	X'00000931'
MQRC_UOW_ENLISTMENT_ERROR	2354	X'00000932'
MQRC_UOW_MIX_NOT_SUPPORTED	2355	X'00000933'
MQRC_WXP_ERROR	2356	X'00000934'
MQRC_CURRENT_RECORD_ERROR	2357	X'00000935'
MQRC_NEXT_OFFSET_ERROR	2358	X'00000936'
MQRC_NO_RECORD_AVAILABLE	2359	X'00000937'
MQRC_OBJECT_LEVEL_INCOMPATIBLE	2360	X'00000938'
MQRC_APPL_FIRST	900	X'00000384'
MQRC_APPL_LAST	999	X'000003E7'

MQRFH_* (Rules and formatting header flags)

See the *Flags* field described in "Chapter 15. MQRFH - Rules and formatting header" on page 237.

MQRFH_NONE	0	X'00000000'

MQRFH_* (Rules and formatting header length)

See the *StrucLength* field described in "Chapter 15. MQRFH - Rules and formatting header" on page 237.

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 15. MQRFH - Rules and formatting header" on page 237.

MQRFH_STRUC_ID	'RFHb'

For the C programming language, the following array version is also defined:

MQRFH_STRUC_ID_ARRAY	'R','F','H','b'

MQRFH_* (Rules and formatting header version)

See the *Version* field described in "Chapter 15. MQRFH - Rules and formatting header" on page 237.

MQRFH_VERSION_1	1	X'00000001'
MQRFH_VERSION_2	2	X'00000002'

MQRL_* (Returned length)

See the *ReturnedLength* field described in "Chapter 7. MQGMO - Get-message options" on page 81.

MQRL_UNDEFINED	-1	X'FFFFFFFF'

MQRMH_* (Reference message header structure identifier)

See the *StrucId* field described in "Chapter 17. MQRMH - Reference message header" on page 251.

MQRMH_STRUC_ID	'RMHb'

For the C programming language, the following array version is also defined:

MQRMH_STRUC_ID_ARRAY	'R','M','H','b'

MQRMH_* (Reference message header version)

See the *Version* field described in "Chapter 17. MQRMH - Reference message header" on page 251.

MQRMH_VERSION_1	1	X'00000001'
MQRMH_CURRENT_VERSION	1	X'00000001'

MQRMHF_* (Reference message header flags)

See the *Flags* field described in "Chapter 17. MQRMH - Reference message header" on page 251.

MQRMHF_NOT_LAST	0	X'00000000'
MQRMHF_LAST	1	X'00000001'

MQRO_* (Report options)

See the *Report* field described in "Chapter 9. MQMD - Message descriptor" on page 125.

MQRO_NONE	0	X'00000000'
MQRO_DEAD_LETTER_Q	0	X'00000000'
MQRO_COPY_MSG_ID_TO_CORREL_ID	0	X'00000000'
MQRO_NEW_MSG_ID	0	X'00000000'
MQRO_PAN	1	X'00000001'
MQRO_EXCEPTION_WITH_FULL_DATA	117440512	X'07000000'
MQRO_PASS_MSG_ID	128	X'00000080'
MQRO_DISCARD_MSG	134217728	X'08000000'
MQRO_COD_WITH_FULL_DATA	14336	X'00003800'
MQRO_EXPIRATION_WITH_FULL_DATA	14680064	X'00E00000'
MQRO_EXCEPTION	16777216	X'01000000'
MQRO_COA_WITH_FULL_DATA	1792	X'00000700'
MQRO_NAN	2	X'00000002'
MQRO_COD	2048	X'00000800'
MQRO_EXPIRATION	2097152	X'00200000'
MQRO_COA	256	X'00000100'
MQRO_EXCEPTION_WITH_DATA	50331648	X'03000000'
MQRO_COD_WITH_DATA	6144	X'00001800'
MQRO_EXPIRATION_WITH_DATA	6291456	X'00600000'
MQRO_PASS_CORREL_ID	64	X'00000040'
MQRO_COA_WITH_DATA	768	X'00000300'

MQRO_* (Report-options masks)

See "Appendix E. Report options and message flags" on page 593.

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 39. Attributes for queues" on page 427.

MQSCO_Q_MGR	1	X'00000001'
MQSCO_CELL	2	X'00000002'

MQSEG_* (Segmentation)

See the *Segmentation* field described in "Chapter 7. MQGMO - Get-message options" on page 81.

MQSEG_INHIBITED	'b'
MQSEG_ALLOWED	'A'

MQSID_* (Security identifier)

See the *AlternateSecurityId* field described in "Chapter 11. MQOD - Object descriptor" on page 193.

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 11. MQOD - Object descriptor" on page 193.

MQSIDT_NONE	X'00'
MQSIDT_NT_SECURITY_ID	X'01'

MQSP_* (Syncpoint)

See the *SyncPoint* attribute described in "Chapter 42. Attributes for the queue manager" on page 469.

MQSP_NOT_AVAILABLE	0	X'00000000'
MQSP_AVAILABLE	1	X'00000001'

MQSS_* (Segment status)

See the *SegmentStatus* field described in "Chapter 7. MQGMO - Get-message options" on page 81.

MQSS_NOT_A_SEGMENT	'b'
MQSS_LAST_SEGMENT	'L'
MQSS_SEGMENT	'S'

MQTC_* (Trigger control)

See the *TriggerControl* attribute described in "Chapter 39. Attributes for queues" on page 427.

MQTC_OFF	0	X'00000000'
MQTC_ON	1	X'00000001'

MQTM_* (Trigger message structure identifier)

See the *StrucId* field described in "Chapter 19. MQTM - Trigger message" on page 265.

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 19. MQTM - Trigger message" on page 265.

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 20. MQTMC2 - Trigger message 2 (character format)" on page 273.

MQTMC_STRUC_ID	'TMCb'

For the C programming language, the following array version is also defined:

MQTMC_STRUC_ID_ARRAY	'T','M','C','b'

MQTMC_* (Trigger message character format structure version)

See the *Version* field described in "Chapter 20. MQTMC2 - Trigger message 2 (character format)" on page 273.

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 39. Attributes for queues" on page 427.

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 39. Attributes for queues" on page 427.

MQUS_NORMAL	0	X'00000000'
MQUS_TRANSMISSION	1	X'00000001'

MQWI_* (Wait interval)

See the *WaitInterval* field described in "Chapter 7. MQGMO - Get-message options" on page 81.

MQWI_UNLIMITED	-1	X'FFFFFFFF'

MQWIH_* (Workload information header flags)

See the *Flags* field described in "Chapter 21. MQWIH - Work information header" on page 279.

MQWIH_NONE	0	X'00000000'

MQWIH_* (Workload information header structure length)

See the *StrucLength* field described in "Chapter 21. MQWIH - Work information header" on page 279.

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 21. MQWIH - Work information header" on page 279.

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 21. MQWIH - Work information header" on page 279.

MQWIH_VERSION_1	1	X'00000001'
MQWIH_CURRENT_VERSION	1	X'00000001'

List of constants

MQXC_* (Exit command identifier)

See the *ExitCommand* field described in "Chapter 22. MQXP - Exit parameter block (OS/390 only)" on page 285.

MQXC_MQOPEN	1	X'00000001'
MQXC_MQCMIT	10	X'0000000A'
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'

MQXCC_* (Exit response)

See the *ExitResponse* field described in "Chapter 22. MQXP - Exit parameter block (OS/390 only)" on page 285.

MQXCC_SUPPRESS_FUNCTION	-1	X'FFFFFFFF'
MQXCC_SKIP_FUNCTION	-2	X'FFFFFFFE'
MQXCC_OK	0	X'00000000'

MQXDR_* (Data-conversion-exit response)

See the *ExitResponse* field described in "MQDXP – Data-conversion exit parameter" on page 607.

MQXDR_OK	0	X'00000000'
MQXDR_CONVERSION_FAILED	1	X'00000001'

MQXP_* (Exit parameter block structure identifier)

See the *StrucId* field described in "Chapter 22. MQXP - Exit parameter block (OS/390 only)" on page 285.

MQXP_STRUC_ID	'XPbb'

For the C programming language, the following array version is also defined:

MQXP_STRUC_ID_ARRAY	'X','P','b','b'

MQXP_* (Exit parameter block version)

See the *Version* field described in "Chapter 22. MQXP - Exit parameter block (OS/390 only)" on page 285.

MQXP_VERSION_1	1	X'00000001'

MQXQH_* (Transmission queue header structure identifier)

See the *StrucId* field described in "Chapter 23. MQXQH - Transmission queue header" on page 291.

MQXQH_STRUC_ID	'XQHb'

For the C programming language, the following array version is also defined:

MQXQH_STRUC_ID_ARRAY	'X','Q','H','b'

MQXQH_* (Transmission queue header version)

See the *Version* field described in "Chapter 23. MQXQH - Transmission queue header" on page 291.

MQXQH_VERSION_1	1	X'00000001'
MQXQH_CURRENT_VERSION	1	X'00000001'

MQXR_* (Exit reason)

See the *ExitReason* field described in "Chapter 22. MQXP - Exit parameter block (OS/390 only)" on page 285.

MQXR_BEFORE	1	X'00000001'
MQXR_AFTER	2	X'00000002'

MQXT_* (Exit identifier)

See the *ExitId* field described in "Chapter 22. MQXP - Exit parameter block (OS/390 only)" on page 285.

MQXT_API_CROSSING_EXIT	1	X'00000001'

MQXUA_* (Exit user area)

See the *ExitUserArea* field described in "Chapter 22. MQXP - Exit parameter block (OS/390 only)" on page 285.

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'

List of constants

Appendix C. Rules for validating MQI options

This appendix lists the situations that produce an MQRC_OPTIONS_ERROR reason code from an 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).

© Copyright IBM Corp. 1994, 2000

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 MQMD (see page 133).

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

© Copyright IBM Corp. 1994, 2000

- 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 IEEE[4] 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:

Encoding	Value to use
Binary integer	MQENC_INTEGER_MASK
Packed-decimal integer	MQENC_DECIMAL_MASK
Floating point	MQENC_FLOAT_MASK

4. The Institute of Electrical and Electronics Engineers

Call the value A.

2. Combine the *Encoding* field with A using the bitwise AND operation; call the result B.
3. 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:

1. Select a value from the following table, according to the encoding required:

Encoding required	Value to use
Binary integer	1
Packed-decimal integer	16
Floating point	256

Call the value A.

2. Divide the value of the *Encoding* field by A; call the result B.
3. Divide B by 16; call the result C.
4. Multiply C by 16 and subtract from B; call the result D.
5. Multiply D by A; call the result E.
6. 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 81.

Table 81. Summary of encodings for machine architectures

Machine architecture	Binary integer encoding	Packed-decimal integer encoding	Floating-point encoding
AS/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 9. MQMD - Message descriptor" on page 125). 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

© Copyright IBM Corp. 1994, 2000

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:

Report type	Value to use
COA	MQRO_COA_WITH_FULL_DATA
COD	MQRO_COD_WITH_FULL_DATA
Exception	MQRO_EXCEPTION_WITH_FULL_DATA
Expiration	MQRO_EXPIRATION_WITH_FULL_DATA

   Call the value `A`.

   On OS/390, 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:

Report type	Value to use
COA	MQRO_COA
COD	MQRO_COD
Exception	MQRO_EXCEPTION
Expiration	MQRO_EXPIRATION

   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
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.

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 601.
- Special considerations for the conversion of report messages; see "Conversion of report messages" on page 605.
- The parameters passed to the data-conversion exit; see "MQ_DATA_CONV_EXIT - Data conversion exit" on page 620.
- A call that can be used from the exit in order to convert character data between different representations; see "MQXCNVC - Convert characters" on page 613.
- The data-structure parameter which is specific to the exit; see "MQDXP – Data-conversion exit parameter" on page 607.

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

© Copyright IBM Corp. 1994, 2000

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
MQFMT_COMMAND_1
MQFMT_COMMAND_2
MQFMT_DEAD_LETTER_HEADER
MQFMT_DIST_HEADER
MQFMT_EVENT
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_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.

- 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

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. Next in precedence is the following reason:
      MQRC_FORMAT_ERROR
   c. 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_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_CCSID_ERROR or MQRC_TARGET_CCSID_ERROR, as appropriate.

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).

- 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

Product-sensitive programming interface

The following table summarizes the fields in the structure.

Table 82. Fields in MQDXP

Field	Description	Page
StrucId	Structure identifier	612
Version	Structure version number	612
AppOptions	Application options	608
Encoding	Numeric encoding required by application	609
CodedCharSetId	Character set required by application	608
DataLength	Length in bytes of message data	608
CompCode	Completion code	608
Reason	Reason code qualifying *CompCode*	610
ExitResponse	Response from exit	609
Hconn	Connection handle	610

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.

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 then 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

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_CCSID_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_CCSID_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).

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 {
  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 */
} MQDXP;
```

COBOL declaration (AS/400 only)

```
**   MQDXP structure
  10 MQDXP.
**    Structure identifier
   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 (OS/390 only)

```
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  Length of structure
                         ORG  MQDXP
MQDXP_AREA               DS   CL(MQDXP_LENGTH)
```

MQXCNVC - Convert characters

The MQXCNVC call converts characters from one character set to another.

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.

Hconn (MQHCONN) – input

Connection handle.

This handle represents the connection to the queue manager. It 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 AS/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 actual 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 actual character set specified on the call, and the default character set.

The default character set is specified by means of 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, AS/400, Sun Solaris, Windows NT.

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.

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, AS/400, Sun Solaris, Windows NT.

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.

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 591 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
   Factor for source encoding

   MQDCC_TARGET_ENC_FACTOR
   Factor for 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, OS/390, OS/2, AS/400, Sun Solaris, Windows NT.

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).

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 489.

C invocation

```
MQXCNVC (Hconn, Options, SourceCCSID, SourceLength, SourceBuffer,
        TargetCCSID, TargetLength, TargetBuffer, &DataLength,
        &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;            /* Connection handle */
MQLONG   Options;          /* Options that control the action of
                              MQXCNVC */
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 (AS/400 only)

```
     CALL 'MQXCNVC' 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 MQXCNVC
 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 (OS/390 only)

```
         CALL MQXCNVC,(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 MQXCNVC
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 623).

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 607 for details of the fields in this structure.

MsgDesc (MQMD) – input/output

Message descriptor.

On input to the exit, this is the message descriptor that would be returned to the application if no conversion were performed. It therefore contains the *Format*, *Encoding*, and *CodedCharSetId* of the unconverted message contained in *InBuffer*.

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, AS/400, Sun Solaris, Windows NT. 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*.

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
- 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.

     In this case, the *Reason* field in the *DataConvExitParms* parameter on input to the exit will have the value MQRC_TRUNCATED_MSG_ACCEPTED.
   - 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 605 for more details).

     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 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 */
     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 OS/390, 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);
```

Declare the parameters 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 uncon-
                               verted message */
MQLONG  OutBufferLength;    /* Length in bytes of OutBuffer */
MQBYTE  OutBuffer[n];       /* Buffer containing the converted
                               message */
```

COBOL invocation (AS/400 only)

```
     CALL 'exitname' USING DATACONVEXITPARMS, MSGDESC,
                           INBUFFERLENGTH, INBUFFER, OUTBUFFERLENGTH,
                           OUTBUFFER.
```

Declare the parameters 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 (OS/390 only)

```
     CALL EXITNAME,(DATACONVEXITPARMS,MSGDESC,INBUFFERLENGTH,INBUFFER, X
           OUTBUFFERLENGTH,OUTBUFFER)
```

Declare the parameters 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 (Tandem NSK only)

For backwards compatibility with MQSeries for Tandem NSK, Version 1.5.1, the signal mode of message-arrival notification is supported. 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.

© Copyright IBM Corp. 1994, 2000

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 Tandem 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.

Appendix H. Code page conversion tables

Each of the tables shows the conversion support for the characters used by one language.

Some of the coded character set identifiers (CCSIDs) are used by many languages, for example CCSID 819 (ISO8859-1 Western European), and appear in many tables. Other CCSIDs, for example CCSID 273 (German EBCDIC), appear in only one table.

The following terms are used in the tables:

ISO Indicates that the CCSID is for an ISO 8859 codeset

pc-A Indicates in the AIX and NCR rows that the CCSID is an IBM defined CCSID used in AIX, AT&T, and OS/2

-8 Indicates in the HP-UX rows that the CCSID is for the HP-UX defined codeset *roman8*

MVS Indicates MQSeries for OS/390

NCR Indicates MQSeries for AT&T GIS UNIX

NT Indicates MQSeries for Windows NT

Solaris
Indicates MQSeries for Sun Solaris

SINIX, DC/OSx
Indicates MQSeries for SINIX and DC/OSx

DEC-OVMS
Indicates MQSeries for Compaq (DIGITAL) OpenVMS

Tru64 Indicates MQSeries for Compaq Tru64 UNIX

Tandem
Indicates MQSeries for Tandem NonStop Kernel, V2.2

The following codes are used in the tables:

Y Conversion at target supported going to and from source

y No conversion is required because the different MQSeries products are operating in the same CCSID

The default for data conversion is for the conversion to be performed at the target (receiving) system.

Where a cell in a table is blank, conversion is not supported by the target product.

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. To determine if the source product supports the conversion, read the relevant table with source and target reversed. If conversion is shown as supported, it is possible to do conversion in the source product.

Notes:

1. Conversion for MQSeries client information takes place in the server, so the server must support conversion from the client CCSID to the server CCSID.

© Copyright IBM Corp. 1994, 2000

2. The numbered notes in the main tables: for example (2) have the same text in each table. Not all the numbers are used in each table.
3. The conversion tables may include support added by CSD/PTF to the latest version of MQSeries. 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 83 for a cross reference between some of the CCSID numbers and some industry codeset names.

Codeset names and CCSIDs

Table 83. Codeset names and CCSIDs

Codeset names	CCSIDs
ISO 8859-1	819
ISO 8859-2	912
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

MQSeries for OS390 provides more conversion than is listed in the language specific tables. A complete list of conversions provided is shown in Table 116 on page 666.

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 682.

Where OS/400 operating system levels are indicated these should be at the following PTF levels or later:
V3R2 SF43993
V3R6 SF43804
V3R7 SF38997
V4R1 SF44021
V4R2 SF49531
V4R3 SF50177

How to read the tables

There is one row for each MQ product. The data in the row shows which conversions this product supports. The first column shows the product. The native CCSID column shows the CCSID used by the product for the national language of the table. The remaining columns show which CCSIDs the product can convert to and from.

Code page conversion tables

The following tables show the conversion support, between the source and target systems, for each of the national languages.

Table 84. Conversion support: US ENGLISH

Product ▼	Native CCSID	MVS, OS/400			OS/2, NCR, NT	AIX, HP-UX, NCR, Solaris, DEC-OVMS, Tandem, Tru64		AIX, NCR, NT, Tru64		HP-UX	Windows client		Apple client
		pre- euro	with euro	with euro	pre- euro	pre- euro	with euro	pre- euro	with euro	pre- euro	pre- euro	with euro	pre- euro
		37	1140	924	437	819	923	850	858	1051	1252	5348	1275
MVS	37	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	1140	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	924	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y
OS/400	37	*y*	Y		Y	Y		Y		Y	Y		Y
	1140	Y	*y*			Y	Y				Y		
	924			*y*			Y						
OS/2	437	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	858	Y	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y
AIX	850 (pc-A)	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	5348 ('1252')	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
HP-UX (ISO)	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	1051 (roman8)	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
NCR	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	437 (pc-A)	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	850 (pc-A)	Y	Y	Y	Y	Y	Y	*y*	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
NT	437	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	850	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	5348 ('1252')	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
Solaris	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
SINIX, DC/OSx	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
DEC-OVMS	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
Tandem	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
Tru64	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
	850	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y

Table 85. Conversion support: GERMAN

Product ▼		MVS, OS/400			NCR, NT	AIX, HP-UX, NCR, Solaris, DEC-OVMS, Tandem, Tru64		OS/2, AIX, NCR, NT		HP-UX	Windows client		Apple client
	Native CCSID	pre- euro	with euro	with euro	pre- euro	pre- euro	with euro	pre- euro	with euro	pre- euro	pre- euro	with euro	pre- euro
		273	1141	924	437	819	923	850	858	1051	1252	5348	1275
MVS	273	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	1141	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	924	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y
OS/400	273	*y*	Y		Y	Y		Y		Y	Y		
	1141	Y	*y*			Y	Y				Y		
	924			*y*			Y						
OS/2	850	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	858	Y	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y
AIX	850 (pc-A)	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	5348 ('1252')	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
HP-UX (ISO)	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	1051 (roman8)	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
NCR	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	437 (pc-A)	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	850 (pc-A)	Y	Y	Y	Y	Y	Y	*y*	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
NT	437	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	850	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	5348 ('1252')	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
Solaris	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
SINIX, DC/OSx	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
DEC-OVMS	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
Tandem	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
Tru64	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y

Table 86. Conversion support: DANISH and NORWEGIAN

Product ▼		MVS, OS/400			AIX, HP-UX, NCR, Solaris, DEC-OVMS, Tandem, Tru64		OS/2, AIX, NCR, NT		OS/2, NCR, NT	HP-UX	Windows client		Apple client
	Native CCSID	pre- euro	with euro	with euro	pre- euro	with euro	pre- euro	with euro	pre- euro	pre- euro	pre- euro	with euro	pre- euro
		277	1142	924	819	923	850	858	865	1051	1252	5348	1275
MVS	277	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	1142	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	924	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y
OS/400	277	*y*	Y		Y		Y		Y	Y	Y		
	1142	Y	*y*		Y	Y					Y		
	924			*y*		Y							
OS/2	850	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
	865	Y	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y
	858	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
AIX	850 (pc-A)	Y	Y	Y	Y	Y	*y*	Y		Y	Y	Y	Y
	819 (ISO)	Y	Y	Y	*y*	Y	Y	Y		Y	Y	Y	Y
	5348 ('1252')	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
HP-UX (ISO)	819 (ISO)	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	1051 (roman8)	Y	Y	Y	Y	Y	Y	Y		*y*	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
NCR	819 (ISO)	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	850 (pc-A)	Y	Y	Y	Y	Y	*y*	Y	Y	Y		Y	
	865 (pc-A)	Y	Y	Y	Y	Y	Y	Y	*y*			Y	
	923 (ISO-15)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
NT	850	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	865	Y	Y	Y	Y	Y	Y	Y	*y*		Y	Y	
	5348 ('1252')	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
Solaris	819	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
SINIX, DC/OSx	819	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
DEC-OVMS	819	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
Tandem	819	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
Tru64	819	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y

Code page conversion tables

Table 87. Conversion support: FINNISH and SWEDISH

Product ▼		MVS, OS/400			NCR, NT	AIX, HP-UX, NCR, Solaris, DEC-OVMS, Tandem, Tru64		OS/2, AIX, NCR, NT		OS/2, NT	HP-UX	Windows client		Apple client
	Native CCSID	pre- euro	with euro	with euro	pre- euro	pre- euro	with euro	pre- euro	with euro	pre- euro	pre- euro	pre- euro	with euro	pre- euro
		278	1143	924	437	819	923	850	858	865	1051	1252	5348	1275
MVS	278	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	1143	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	924	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
OS/400	278	*y*	Y		Y	Y		Y		Y	Y	Y		
	1143	Y	*y*			Y	Y					Y		
	924			*y*			Y							
OS/2	850	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
	865	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y
	858	Y	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
AIX	850 (pc-A)	Y	Y	Y	Y	Y	Y	*y*	Y		Y	Y	Y	Y
	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y		Y	Y	Y	Y
	5348 ('1252')	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
HP-UX (ISO)	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	1051 (roman8)	Y	Y	Y	Y	Y	Y	Y	Y		*y*	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
NCR	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	437 (pc-A)	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y		Y	
	850 (pc-A)	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
NT	437	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y
	850	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
	865	Y	Y	Y	Y	Y	Y	Y	Y	*y*		Y	Y	
	5348 ('1252')	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
Solaris	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
SINIX, DC/OSx	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
DEC-OVMS	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
Tandem	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
Tru64	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y

Table 88. Conversion support: ITALIAN

Product ▼		MVS, OS/400			NCR, NT	AIX, HP-UX, NCR, Solaris, DEC-OVMS, Tandem, Tru64		OS/2, AIX, NCR, NT		HP-UX	Windows client		Apple client
	Native CCSID	pre- euro	with euro	with euro	pre- euro	pre- euro	with euro	pre- euro	with euro	pre- euro	pre- euro	with euro	pre- euro
		280	1144	924	437	819	923	850	858	1051	1252	5348	1275
MVS	280	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	1144	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	924	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y
OS/400	280	*y*	Y		Y	Y		Y		Y	Y		
	1144	Y	*y*			Y	Y				Y		
	924			*y*			Y						
OS/2	850	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	858	Y	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y
AIX	850 (pc-A)	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	5348 (’1252’)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
HP-UX (ISO)	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	1051 (roman8)	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
NCR	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	437 (pc-A)	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	850 (pc-A)	Y	Y	Y	Y	Y	Y	*y*	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
NT	437	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	850	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	5348 (’1252’)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
Solaris	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
SINIX, DC/OSx	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
DEC-OVMS	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
Tandem	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
Tru64	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y

Table 89. Conversion support: SPANISH

Product ▼		MVS, OS/400			NCR, NT	AIX, HP-UX, NCR, Solaris, DEC-OVMS, Tandem, Tru64		OS/2, AIX, NCR, NT		HP-UX	Windows client		Apple client
	Native CCSID	pre- euro	with euro	with euro	pre- euro	pre- euro	with euro	pre- euro	with euro	pre- euro	pre- euro	with euro	pre- euro
		284	1145	924	437	819	923	850	858	1051	1252	5348	1275
MVS	284	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	1145	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	924	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y
OS/400	284	*y*	Y		Y	Y		Y		Y	Y		
	1145	Y	*y*			Y	Y				Y		
	924			*y*			Y						
OS/2	850	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	858	Y	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y
AIX	850 (pc-A)	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	5348 ('1252')	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
HP-UX (ISO)	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	1051 (roman8)	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
NCR	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	437 (pc-A)	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	850 (pc-A)	Y	Y	Y	Y	Y	Y	*y*	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
NT	437	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	850	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	5348 ('1252')	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
Solaris	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
SINIX, DC/OSx	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
DEC-OVMS	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
Tandem	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
Tru64	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y

Table 90. Conversion support: UK ENGLISH / GAELIC

Product ▼		MVS, OS/400			NCR, NT	AIX, HP-UX, NCR, Solaris, DEC-OVMS, Tandem, Tru64		OS/2, AIX, NCR, NT		HP-UX	Windows client		Apple client
	Native CCSID	pre- euro	with euro	with euro	pre- euro	pre- euro	with euro	pre- euro	with euro	pre- euro	pre- euro	with euro	pre- euro
		285	1146	924	437	819	923	850	858	1051	1252	5348	1275
MVS	285	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	1146	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	924	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y
OS/400	285	*y*	Y		Y	Y		Y		Y	Y		
	1146	Y	*y*			Y	Y				Y		
	924			*y*			Y						
OS/2	850	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	858	Y	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y
AIX	850 (pc-A)	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	5348 ('1252')	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
HP-UX (ISO)	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	1051 (roman8)	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
NCR	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	437 (pc-A)	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	850 (pc-A)	Y	Y	Y	Y	Y	Y	*y*	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
NT	437	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	850	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	5348 ('1252')	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
Solaris	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
SINIX, DC/OSx	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
DEC-OVMS	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
Tandem	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
Tru64	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y

Table 91. Conversion support: FRENCH

Product ▼		MVS, OS/400			NCR, NT	AIX, HP-UX, NCR, Solaris, DEC-OVMS, Tandem, Tru64		OS/2, AIX, NCR, NT		HP-UX	Windows client		Apple client
	Native CCSID	pre- euro	with euro	with euro	pre- euro	pre- euro	with euro	pre- euro	with euro	pre- euro	pre- euro	with euro	pre- euro
		297	1147	924	437	819	923	850	858	1051	1252	5348	1275
MVS	297	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	1147	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	924	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y
OS/400	297	*y*	Y		Y	Y		Y		Y	Y		
	1147	Y	*y*			Y	Y	Y			Y		
	924			*y*			Y						
OS/2	850	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	858	Y	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y
AIX	850 (pc-A)	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	5348 ('1252')	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
HP-UX (ISO)	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	1051 (roman8)	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
NCR	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	437 (pc-A)	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	850 (pc-A)	Y	Y	Y	Y	Y	Y	*y*	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
NT	437	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	850	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	5348 ('1252')	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
Solaris	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
SINIX, DC/OSx	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
DEC-OVMS	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
Tandem	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
Tru64	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y

Table 92. Conversion support: MULTILINGUAL

Product ▼		MVS, OS/400			NCR, NT	AIX, HP-UX, NCR, Solaris, DEC-OVMS, Tandem, Tru64		OS/2, AIX, NCR, NT		HP-UX	Windows client		Apple client
	Native CCSID	pre- euro	with euro	with euro	pre- euro	pre- euro	with euro	pre- euro	with euro	pre- euro	pre- euro	with euro	pre- euro
		500	1148	924	437	819	923	850	858	1051	1252	5348	1275
MVS	500	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	1148	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	924	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y
OS/400	500	*y*	Y		Y	Y		Y		Y	Y		Y
	1148	Y	*y*			Y	Y	Y			Y		
	924	Y		*y*			Y						
OS/2	850	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	858	Y	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y
AIX	850 (pc-A)	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	5348 ('1252')	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
HP-UX (ISO)	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	1051 (roman8)	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
NCR	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	437 (pc-A)	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	850 (pc-A)	Y	Y	Y	Y	Y	Y	*y*	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
NT	437	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	850	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	5348 ('1252')	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
Solaris	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
SINIX, DC/OSx	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
DEC-OVMS	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
Tandem	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
Tru64	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y

Table 93. Conversion support: PORTUGUESE

Product | ▾		MVS, OS/400			OS/400	AIX, HP-UX, NCR, Solaris, DEC-OVMS, Tandem, Tru64		OS/2, AIX, NCR, NT		OS/2, NCR, NT	HP-UX	Windows client		Apple client
	Native CCSID	pre- euro	with euro	with euro	pre- euro	pre- euro	with euro	pre- euro	with euro	pre- euro	pre- euro	pre- euro	with euro	pre- euro
		500	1140	924	37	819	923	850	858	860	1051	1252	5348	1275
MVS	500	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	1140	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	924	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
OS/400	37	Y	Y	Y	*y*	Y		Y		Y	Y	Y		
	500	*y*	Y		Y	Y		Y		Y	Y	Y		
	1140	Y	*y*		Y	Y	Y					Y		
	924			*y*			Y							
OS/2	850	Y	Y	Y	Y	Y	Y	*y*	Y		Y	Y	Y	Y
	860	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y
	858	Y	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
AIX	850 (pc-A)	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	5348 ('1252')	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
HP-UX (ISO)	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	1051 (roman8)	Y	Y	Y	Y	Y	Y	Y	Y		*y*	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
NCR	819 (ISO)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	850 (pc-A)	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y		Y	
	860 (pc-A)	Y	Y	Y	Y	Y	Y	Y	Y	*y*			Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
NT	850	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
	860	Y	Y	Y	Y	Y	Y	Y	Y	*y*		Y	Y	
	5348 ('1252')	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
Solaris	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
SINIX, DC/OSx	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
DEC-OVMS	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
Tandem	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
Tru64	819	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y

Table 94. Conversion support: ICELANDIC

Product ▼		MVS, OS/400			AIX, HP-UX, NCR, Solaris, DEC-OVMS, Tandem, Tru64		OS/2, AIX, NCR, NT		OS/2, NCR, NT	HP-UX	Windows client		Apple client
	Native CCSID	pre- euro	with euro	with euro	pre- euro	with euro	pre- euro	with euro	pre- euro	pre- euro	pre- euro	with euro	pre- euro
		871	1149	924	819	923	850	858	861	1051	1252	5348	1275
MVS	871	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	1149	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	924	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y
OS/400	871	*y*	Y		Y		Y		Y	Y	Y		
	1149	Y	*y*		Y	Y					Y		
	924			*y*		Y							
OS/2	850	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
	861	Y	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y
	858	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
AIX	850 (pc-A)	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y
	819 (ISO)	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	5348 ('1252')	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
HP-UX (ISO)	819 (ISO)	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	1051 (roman8)	Y	Y	Y	Y	Y	Y	Y		*y*	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
NCR	819 (ISO)	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	850 (pc-A)	Y	Y	Y	Y	Y	*y*	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
NT	850	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
	861	Y	Y	Y	Y	Y	Y	Y	*y*		Y	Y	
	5348 ('1252')	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
Solaris	819	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
SINIX, DC/OSx	819	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
DEC-OVMS	819	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
Tandem	819	Y	Y	Y	*y*	Y	Y	Y	Y	Y		Y	
	923 (ISO-15)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
Tru64	819	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	923 (ISO-15)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y

Table 95. Conversion support: EASTERN EUROPEAN Languages

Product ▼		OS/2, NT	MVS, OS/400	AIX, HP-UX, NCR, Solaris, DEC-OVMS, Tandem, Tru64	Windows client		Apple client Eastern European	Apple client Croatian	Apple client Romanian
	Native CCSID				pre-euro	with euro			
		852	870	912	1250	5346	1282	1284	1285
MVS	870	Y	*y*	Y	Y	Y	Y		
OS/400	870	Y	*y*	Y	Y		Y		
OS/2	852	*y*	Y	Y	Y	Y	Y	Y	Y
AIX (ISO)	912	Y	Y	*y*	Y	Y	Y	Y	Y
HP-UX (ISO)	912	Y	Y	*y*	Y	Y	Y		
NCR (ISO)	912	Y	Y	*y*					
NT	852	*y*	Y	Y	Y	Y	Y		
	5346	Y	Y	Y	Y	*y*	Y	Y	Y
Solaris	912	Y	Y	*y*	Y	Y	Y		
SINIX, DC/OSx	912	Y	Y	*y*	Y	Y	Y		
DEC-OVMS	912	Y	Y	*y*	Y	Y	Y		
Tandem	912	Y	Y	*y*		Y			
Tru64	912	Y	Y	*y*	Y	Y	Y		

Notes:

- The typical languages which use these CCSIDS include Albanian, Croatian, Czech, Hungarian, Polish, Romanian, Serbian, Slovakian, and Sloven.

Table 96. Conversion support: CYRILLIC

Product ▼		OS/2, NT	OS/2, NT	Solaris	OS/400	AIX, HP-UX, NCR, Solaris, DEC-OVMS, Tandem, Tru64	MVS, OS/400	OS/2, NT	Windows client		Apple client
	Native CCSID								pre-euro	with euro	
		855	866	878	880	915	1025	1131	1251	5347	1283
MVS	1025	Y	Y	Y	Y	Y	*y*	Y	Y	Y	Y
OS/400	880				*y*	Y	Y		Y		Y
	1025	Y	Y		Y	Y	*y*	Y	Y		Y
OS/2	855	*y*	Y	Y	Y	Y	Y	Y	Y	Y	Y
	866	Y	*y*	Y	Y	Y	Y	Y	Y	Y	Y
	1131	Y	Y	Y	Y	Y	Y	*y*	Y	Y	Y
AIX (ISO)	915	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
HP-UX (ISO)	915	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
NCR (ISO)	915	Y	Y		Y	*y*	Y			Y	
NT	855	*y*	Y	Y	Y	Y	Y		Y	Y	Y
	866	Y	*y*	Y	Y	Y	Y		Y	Y	Y
	1131			Y		Y	Y	*y*	Y	Y	
	5347	Y	Y	Y	Y	Y	Y	Y	Y	*y*	Y
Solaris	878 (koi8-r)	Y	Y	*y*	Y	Y	Y	Y	Y	Y	Y
	915 (iso8859-5)	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y
SINIX, DC/OSx	915	Y	Y		Y	*y*	Y		Y	Y	Y
DEC-OVMS	915	Y	Y		Y	*y*	Y		Y	Y	Y
Tandem	915	Y	Y		Y	*y*	Y			Y	
Tru64	915	Y	Y	Y	Y	*y*	Y	Y	Y	Y	Y

Notes:

- The typical languages which use these CCSIDS include Byelorussia (Belarus), Bulgarian, Macedonian, Russian, and Serbian.

Table 97. Conversion support: ESTONIAN

Product ▼		OS/2, AIX, HP-UX, NT, Solaris, Tru64	MVS, OS/400	Windows client	
	Native CCSID			pre-euro	with euro
		922	1122	1257	5353
MVS	1122	**Y**	*y*	**Y**	**Y**
OS/400	1122	**Y**	*y*	**Y**	
AIX	922	*y*	**Y**	**Y**	**Y**
OS/2	922	*y*	**Y**	**Y**	**Y**
HP-UX	922	*y*	**Y**	**Y**	**Y**
NCR					
NT	922	*y*	**Y**	**Y**	**Y**
	5353	**Y**	**Y**	**Y**	*y*
Solaris	922	*y*	**Y**	**Y**	**Y**
SINIX, DC/OSx					
DEC-OVMS					
Tandem					
Tru64	922	*y*	**Y**	**Y**	**Y**

Table 98. Conversion support: LATVIAN and LITHUANIAN

Product ▼	Native CCSID	OS/2, AIX, HP-UX, NT, Solaris, Tru64	MVS, OS/400	Windows client	
				pre-euro	with euro
		921	1112	1257	5353
MVS	1112	**Y**	*y*	**Y**	**Y**
OS/400	1112	**Y**	*y*	**Y**	
OS/2	921	*y*	**Y**	**Y**	**Y**
AIX	921	*y*	**Y**	**Y**	**Y**
HP-UX	921	*y*	**Y**	**Y**	**Y**
NCR					
NT	921	*y*	**Y**	**Y**	**Y**
	5353	**Y**	**Y**	**Y**	*y*
Solaris	921	*y*	**Y**	**Y**	**Y**
SINIX, DC/OSx					
DEC-OVMS					
Tandem					
Tru64	921	*y*	**Y**	**Y**	**Y**

Table 99. Conversion support: UKRAINIAN

Product ▼		MVS, OS/400	AIX, HP-UX, NT, Solaris, Tru64	OS/2, NT	Windows client	
	Native CCSID				pre-euro	with euro
		1123	1124	1125	1251	5347
MVS	1123	*y*	**Y**	**Y**	**Y**	**Y**
OS/400	1123	*y*		**Y**		
OS/2	1125	**Y**	**Y**	*y*	**Y**	**Y**
AIX	1124	**Y**	*y*	**Y**	**Y**	**Y**
HP-UX	1124	**Y**	*y*	**Y**	**Y**	
NCR						
NT	1124	**Y**	*y*	**Y**	**Y**	**Y**
	1125		**Y**	*y*		**Y**
	5347	**Y**	**Y**	**Y**	**Y**	*y*
Solaris	1124	**Y**	*y*	**Y**	**Y**	**Y**
SINIX, DC/OSx						
DEC-OVMS						
Tandem						
Tru64	1124	**Y**	*y*	**Y**	**Y**	**Y**

Table 100. Conversion support: GREEK

Product ▼		OS/2, AIX, HP-UX, NCR, Solaris, DEC-OVMS, Tandem, Tru64	OS/2, NT	MVS, OS/400	Windows client		Apple client	DOS client
	Native CCSID				pre-euro	with euro		
		813	869	875	1253	5349	1280	737
MVS	875	Y	Y	*y*	Y	Y	Y	Y
OS/400	875	Y	Y	*y*	Y		Y	Y
OS/2	813	*y*	Y	Y	Y	Y	Y	Y
	869	Y	*y*	Y	Y	Y	Y	Y
AIX (ISO)	813	*y*	Y	Y	Y	Y	Y	Y
HP-UX (ISO)	813#	*y*	Y	Y	Y	Y	Y	Y
NCR (ISO)	813	*y*	Y	Y		Y		
NT	869	Y	*y*	Y	Y	Y	Y	Y
	5349	Y	Y	Y	Y	*y*	Y	
Solaris	813	*y*	Y	Y	Y	Y	Y	Y
SINIX, DC/OSx	813	*y*	Y	Y	Y	Y	Y	
DEC-OVMS	813	*y*	Y	Y	Y	Y	Y	
Tandem	813	*y*	Y	Y		Y		
Tru64	813	*y*	Y	Y	Y	Y	Y	Y

Notes:

Only the ISO codeset on HP-UX is supported. The HP-UX proprietary greek8 codeset has no registered CCSID and is not supported.

Table 101. Conversion support: TURKISH

Product ▼		OS/2, NT	AIX, HP-UX, Solaris, DEC-OVMS, Tandem, Tru64	MVS, OS/400	Windows client		Apple client
	Native CCSID				pre-euro	with euro	
		857	920	1026	1254	5350	1281
MVS	1026	Y	Y	*y*	Y	Y	Y
OS/400	1026	Y	Y	*y*	Y		Y
OS/2	857	*y*	Y	Y	Y	Y	Y
AIX (ISO)	920	Y	*y*	Y	Y	Y	Y
HP-UX (ISO)	920#	Y	*y*	Y	Y	Y	Y
NCR							
NT	857	*y*	Y	Y	Y	Y	Y
	5350	Y	Y	Y	Y	*y*	Y
Solaris	920	Y	*y*	Y	Y	Y	Y
SINIX, DC/OSx	920	Y	*y*	Y	Y	Y	Y
DEC-OVMS	920	Y	*y*	Y	Y	Y	Y
Tandem	920	Y	*y*	Y		Y	
Tru64	920	Y	*y*	Y	Y	Y	Y

Notes:
Only the ISO codeset on HP-UX is supported. The HP-UX proprietary turkish8 codeset has no registered CCSID and is not supported.

Table 102. Conversion support: HEBREW

Product ▼		MVS, OS/400		MVS		AIX		OS/2		AIX, HP-UX, Solaris, DEC-OVMS, Tandem, Tru64	Windows NT and Windows client	
	Native CCSID	pre-euro	with euro	pre-euro	with euro	pre-euro	with euro	pre-euro	with euro		pre-euro	with euro
		424	12712	803	4899	856	9048	862	867	916	1255	5351
MVS	424	*y*		**Y**		**Y**		**Y**		**Y**	**Y**	**Y**
	12712		*y*		**Y**		**Y**	**Y**	**Y**			**Y**
	803	**Y**		*y*		**Y**		**Y**		**Y**	**Y**	
	4899		**Y**		*y*		**Y**		**Y**			**Y**
OS/400	424	*y*				**Y#**		**Y**		**Y**	**Y**	
	12712		*y*									
OS/2	862	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	*y*	**Y**	**Y**	**Y**	**Y**
	867	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	*y*	**Y**	**Y**	**Y**
AIX	856 (pc-A)	**Y**		**Y**		*y*		**Y**		**Y**	**Y**	**Y**
	9048 (pc-A)		**Y**		**Y**		*y*		**Y**			**Y**
	916 (ISO)	**Y**		**Y**		**Y**		**Y**		*y*	**Y**	**Y**
HP-UX (ISO)	916§	**Y**		**Y**		**Y**		**Y**		*y*	**Y**	**Y**
NCR												
NT	1255	**Y**		**Y**		**Y**		**Y**		**Y**	*y*	**Y**
	5351	**Y**	**Y**		**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	*y*
Solaris	916	**Y**		**Y**		**Y**		**Y**		*y*	**Y**	**Y**
SINIX, DC/OSx	916	**Y**				**Y**		**Y**	*y*		**Y**	**Y**
DEC-OVMS	916	**Y**				**Y**		**Y**		*y*	**Y**	**Y**
Tandem	916	**Y**				**Y**		**Y**		*y*		**Y**
Tru64	916	**Y**		**Y**		**Y**		**Y**		*y*	**Y**	**Y**

Notes:

Only to/from CCSID 4952 (a variant of 856).

§ Only the ISO codeset on HP-UX is supported. The HP-UX proprietary hebrew8 codeset has no registered CCSID and is not supported.

Table 103. Conversion support: ARABIC

Product ▼	Native CCSID	MVS, OS/400	OS/2, NT	AIX	AIX, HP-UX, Solaris, DEC-OVMS, Tandem, Tru64	Windows client	
	Native CCSID					pre-euro	with euro
		420	864	1046	1089	1256	5352
MVS	420	*y*	Y	Y	Y	Y	Y
OS/400	420	*y*	Y	Y	Y	Y	
OS/2	864	Y	*y*	Y	Y	Y	Y
AIX (pc-A)	1046	Y	Y	*y*	Y	Y	Y
	1089	Y	Y	Y	*y*	Y	Y
HP-UX (ISO)	1089§	Y	Y	Y	*y*	Y	Y
NCR							
NT	864	Y	*y*	Y	Y	Y	Y
	5352	Y	Y	Y	Y	Y	*y*
Solaris	1089	Y	Y	Y	*y*	Y	Y
SINIX, DC/OSx	1089	Y	Y	Y	*y*	Y	Y
DEC-OVMS	1089	Y	Y	Y	*y*	Y	Y
Tandem	1089	Y	Y	Y	*y*		Y
Tru64	1089	Y	Y	Y	*y*	Y	Y

Notes:
§ Only the ISO codeset on HP-UX is supported. The HP-UX proprietary arabic8 codeset has no registered CCSID and is not supported.

Table 104. Conversion support: FARSI

Product ▼		**MVS, OS/400**	**OS/2**
	Native CCSID	**1097**	**1098**
MVS	1097	*y*	**Y**
OS/400	1097	*y*	**Y**
OS/2	1098	**Y**	*y*
AIX	1098(8)	**Y**	*y*
HP-UX	1098(8)	**Y**	*y*
NCR			
NT	1098(8)	**Y**	*y*
Solaris	1098(8)	**Y**	*y*
SINIX, DC/OSx			
DEC-OVMS			
Tandem			
Tru64	1098(8)	**Y**	*y*

Notes:
(8) The native CCSID for these platforms has not been standardized and may change

Table 105. Conversion support: URDU

Product ▼		OS/2, NT	MVS, OS/400	AIX, HP-UX, Solaris, Tru64
	Native CCSID	**868**	**918**	**1006**
MVS	918	**Y**	*y*	**Y**
OS/400	918	**Y**	*y*	
OS/2	868	*y*	**Y**	**Y**
AIX	1006	**Y**	**Y**	*y*
HP-UX	1006	**Y**	**Y**	*y*
NCR				
NT	868	*y*	**Y**	**Y**
Solaris	1006	**Y**	**Y**	*y*
SINIX, DC/OSx				
DEC-OVMS				
Tandem				
Tru64	1006	**Y**	**Y**	*y*

Table 106. Conversion support: THAI

Product ▼		**MVS, OS/400**	**OS/2**
	Native CCSID	**838**	**874**
MVS	838	*y*	**Y**
OS/400	838	*y*	**Y**
OS/2	874	**Y**	*y*
AIX	874(8)	**Y**	*y*
HP-UX	874(8)	**Y**	*y*
NCR			
NT	874(8)	**Y**	*y*
Solaris	874(8)	**Y**	*y*
SINIX, DC/OSx			
DEC-OVMS			
Tandem			
Tru64	874(8)	**Y**	*y*

Notes:
(8) The native CCSID for these platforms has not been standardized and may change

Table 107. Conversion support: LAO

Product ▼		**MVS, OS/400**	**OS/2, NT, AIX, HP-UX, Solaris, Tru64**
	Native CCSID	**1132**	**1133**
MVS	1132	*y*	**Y**
OS/400	1132	*y*	**Y**
OS/2	1133	**Y**	*y*
AIX	1133	**Y**	*y*
HP-UX	1133	**Y**	*y*
NCR			
NT	1133	**Y**	*y*
Solaris	1133	**Y**	*y*
SINIX, DC/OSx			
DEC-OVMS			
Tandem			
Tru64	1133	**Y**	*y*

Table 108. Conversion support: VIETNAMESE

Product ▼		OS/2, AIX, HP-UX, Solaris, Tru64	MVS, OS/400	Windows NT and Windows clients	
	Native CCSID			pre-euro	with euro
		1129	1130	1258	5354
MVS	1130	Y	*y*	Y	Y
OS/400	1130		*y*	Y	
OS/2	1129	*y*	Y	Y	Y
AIX	1129	*y*	Y	Y	Y
HP-UX	1129	*y*	Y	Y	Y
NCR					
NT	1258	Y	Y	*y*	Y
	5354	Y	Y	Y	*y*
Solaris	1129	*y*	Y	Y	Y
SINIX, DC/OSx					
DEC-OVMS					
Tandem					
Tru64	1129	*y*	Y	Y	Y

Table 109. Conversion support: JAPANESE LATIN SBCS

Product ▼		**OS/2, AIX, NT**	**OS/2**	**NT, Solaris, Tru64**	**MVS, OS/400**	**AIX, Solaris, Tru64**
	Native CCSID	**932**	**942**	**943**	**1027**	**954 5050 33722**
MVS	1027				*y*	
OS/400	1027		**Y**	**Y**	*y*	
OS/2	932	*y*	**Y**	**Y**	**Y**	**Y**
	942	**Y**	*y*	**Y**	**Y**	**Y**
AIX	932 (pc-A)	*y*	**Y**	**Y**		**Y**
	5050 33722* (euc)	**Y**	**Y**	**Y**		*y*
HP-UX						
NCR						
NT	932##	*y*	**Y**	**Y**	**Y**	**Y**
	943##	**Y**	**Y**	*y*	**Y**	**Y**
Solaris	5050 (euc)	**Y**	**Y**	**Y**		*y*
	943 (PCK)	**Y**	**Y**	*y*		**Y**
SINIX, DC/OSx						
DEC-OVMS						
Tandem						
Tru64	954 (euc) 33722(deckanji)	**Y**	**Y**	**Y**		*y*
	943 (SJIS)	**Y**	**Y**	*y*		**Y**

Notes:

* 5050 and 33722 are CCSIDs related to base code page 954 = eucJP on AIX. On AIX V4.1 the CCSID reported by the operating system is 33722.

\## Windows NT uses the code page number 932, but this is best represented by the CCSID of 943. However not all platforms of MQSeries support this CCSID. On MQSeries for Windows NT 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.

Table 110. Conversion support: JAPANESE KATAKANA SBCS

Product ▼		MVS, OS/400	OS/2, HP-UX	AIX, NT	NT, Solaris, Tru64	AIX, Solaris, Tru64
	Native CCSID	**290**	**897**	**932**	**943**	**954 5050 33722**
MVS	290	*y*	**Y**			
OS/400	290	*y*	**Y**		**Y**	
OS/2	897	**Y**	*y*	**Y**	**Y**	**Y**
AIX	932 (pc-A)			*y*	**Y**	**Y**
	5050 33722* (euc)			**Y**	**Y**	*y*
HP-UX (kana8)	897	**Y**	*y*			
NCR						
NT	932##	**Y**	**Y**	*y*	**Y**	**Y**
	943##	**Y**	**Y**	**Y**	*y*	**Y**
Solaris	5050 (euc)			**Y**	**Y**	*y*
	943 (PCK)			**Y**	*y*	**Y**
SINIX, DC/OSx						
DEC-OVMS						
Tandem						
Tru64	954 (euc) 33722(deckanji)			**Y**	**Y**	*y*
	943 (SJIS)			**Y**	*y*	**Y**

Notes:

- In addition to the above conversions, MQSeries for AIX, OS2 WARP, HP, NT, Sun Solaris and Tru64 supports conversion from CCSID 897 to CCSIDs 37, 273, 277, 278, 280, 284, 285, 290, 297, 437, 500, 819, 850, 1027, and 1252.

* 5050 and 33722 are CCSIDs related to base code page 954 = eucJP on AIX. On AIX V4.1 the CCSID reported by the operating system is 33722.

Windows NT uses the code page number 932, but this is best represented by the CCSID of 943. However not all platforms of MQSeries support this CCSID. On MQSeries for Windows NT 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.

Table 111. Conversion support: JAPANESE KANJI / LATIN MIXED

Product ▼		OS/2, AIX, HP-UX, DEC-OVMS, Tandem, NT	OS/2	NT, Solaris, Tru64	HP-UX, DEC-OVMS, Tandem, Tru64	MVS, OS/400	MVS, OS/400	AIX, Solaris, Tru64	HP-UX
	Native CCSID	**932**	**942**	**943**	**954**	**1399**	**5035**	**5050 33722**	**5039**
MVS	5035#	**Y**	**Y**	**Y**			*y*		**Y**
	1399	**Y**	**Y**	**Y**		*y*			**Y**
OS/400	5035#	**Y**	**Y**	**Y**			*y*	**Y(9)**	
	1399	**Y(4)**	**Y(4)**	**Y(4)**		*y*			.
OS/2	932	*y*	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	942
	Y	*y*	**Y**	**Y**	**Y**	**Y**	**Y**		
AIX	932 (pc-A)	*y*	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
	5050 33722* (euc)	**Y**	**Y**	**Y**	*y*	**Y**	**Y**	*y*	**Y**
HP-UX	954 (euc)	**Y**			*y*		**Y**	*y*	**Y**
	932 (-15§)	*y*			**Y**		**Y**	**Y**	**Y**
	5039 (-15§)	**Y**			**Y**		**Y**	**Y**	*y*
NCR									
NT	932##	*y*	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
	943##	**Y**	**Y**	*y*	**Y**	**Y**	**Y**	**Y**	**Y**
Solaris	5050 (euc)	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	*y*	**Y**
	943 (PCK)	**Y**	**Y**	*y*	**Y**	**Y**	**Y**	**Y**	**Y**
SINIX, DC/OSx									
DEC-OVMS	932	*y*			**Y**		**Y**	**Y**	
	954	**Y**			*y*		**Y**	*y*	
Tandem	932	*y*			**Y**		**Y**	**Y**	
	954	**Y**			*y*		**Y**	*y*	
Tru64	954 (euc) 33722(deckanji)	**Y**	**Y**	**Y**	*y*		**Y**	*y*	**Y**
	943 (SJIS)	**Y**	**Y**	*y*	**Y**		**Y**	**Y**	**Y**

Notes:

(9) 5050 only, no support for conversion to 33722

(4) Supported on AS/400 V4R5 or later

* 5050 and 33722 are CCSIDs related to base code page 954 = eucJP on AIX. On AIX V4.1 the CCSID reported by the operating system is 33722.

5035 is a CCSID related to code page 939.

-15§ Defined by HP-UX as japan15 and SJIS 932 has a few DBCS characters have different representations in SJIS and 932 so may not be converted correctly if the conversion is performed on a non-HP-UX system. MQSeries for HP-UX supports 5039, the correct CCSID for HP SJIS. A change to `/var/mqm/conv/ccsid.tbl` can be made to change the CCSID used from 932 to 5039

Windows NT uses the code page number 932, but this is best represented by the CCSID of 943. However not all platforms of MQSeries support this CCSID. On MQSeries for Windows NT 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.

Table 112. Conversion support: JAPANESE KANJI / KATAKANA MIXED

Product ▼		OS/2, AIX, HP-UX, DEC-OVMS, Tandem, NT	OS/2	NT, Solaris, Tru64	HP-UX, DEC-OVMS, Tandem, Tru64	MVS	MVS, OS/400	AIX, Solaris, Tru64	HP-UX
	Native CCSID	**932**	**942**	**943**	**954**	**1390**	**5026**	**5050 33722**	**5039**
MVS	5026#	**Y**	**Y**	**Y**			*y*		**Y**
	1390	**Y**	**Y**	**Y**		*y*			**Y**
OS/400	5026#	**Y**	**Y**	**Y**			*y*	**Y(9)**	
OS/2	932	*y*	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
	942	**Y**	*y*	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
AIX	932 (pc-A)	*y*	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
	5050 33722* (euc)	**Y**	**Y**	**Y**	*y*	**Y**	**Y**	*y*	**Y**
HP-UX	954 (euc)	**Y**			*y*		**Y**	*y*	**Y**
	932 (-15§)	*y*			**Y**		**Y**	**Y**	**Y**
	5039 (-15§)	**Y**			**Y**		**Y**	**Y**	*y*
NCR									
NT	932##	*y*	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
	943##	**Y**	**Y**	*y*	**Y**	**Y**	**Y**	**Y**	**Y**
Solaris	5050 (euc)	**Y**	**Y**	**Y**	*y*	**Y**	**Y**	*y*	**Y**
	943 (PCK)	**Y**	**Y**	*y*	**Y**	**Y**	**Y**	**Y**	**Y**
SINIX, DC/OSx									
DEC-OVMS	932 (sjis)	*y*			**Y**		**Y**	**Y**	
	954 (euc)	**Y**			*y*		**Y**	*y*	
Tandem	932 (sjis)	*y*		**Y**	**Y**		**Y**		
	954 (euc)	**Y**			*y*		**Y**	*y*	
Tru64	954 (euc) 33722 (deckanji)	**Y**	**Y**	**Y**	*y*		**Y**	*y*	**Y**
	943 (SJIS)	**Y**	**Y**	*y*	**Y**		**Y**	**Y**	**Y**

Notes:

(9) 5050 only, no support for conversion to 33722

* 5050 and 33722 are CCSIDs related to base code page 954 = eucJP on AIX. On AIX V4.1 the CCSID reported by the operating system is 33722.

5026 is a CCSID related to code page 930. CCSID 5026 is the CCSID reported to the user on OS/400 when the Japanese Katakana (DBCS) feature is selected.

-15§ Defined by HP-UX as japan15 and SJIS 932 has a few DBCS characters have different representations in SJIS and 932 so may not be converted correctly if the conversion is performed on a non-HP-UX system. MQSeries for HP-UX supports 5039, the correct CCSID for HP SJIS. A change to `/var/mqm/conv/ccsid.tbl` can be made to change the CCSID used from 932 to 5039

Windows NT uses the code page number 932, but this is best represented by the CCSID of 943. However not all platforms of MQSeries support this CCSID. On MQSeries for Windows NT 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.

Table 113. Conversion support: KOREAN

Product ▼		**MVS, OS/400**		**OS/2, NT**	**OS/2, NT, Tru64**	**AIX, HP-UX, DEC-OVMS, Tandem, Solaris, Tru64**
	Native CCSID	**933**	**1364**	**949**	**1363**	**970**
MVS	933	*y*	**Y**	**Y**	**Y**	
	1364	**Y**	*y*	**Y**	**Y**	
OS/400	933	*y*	**Y(14)**	**Y**	**Y(14)**	**Y**
	1364	**Y(14)**	*y*	**Y(14)**	**Y(14)**	**Y(14)**
OS/2	949	**Y**	**Y**	*y*	**Y**	**Y**
	1363	**Y**	**Y**	**Y**	*y*	**Y**
AIX	970 (euc)	**Y**	**Y**	**Y**	**Y**	*y*
HP-UX	970 (euc)	**Y**				*y*
NCR						
NT	949	**Y**	**Y**	*y*	**Y**	**Y**
	1363	**Y**	**Y**	**Y**	*y*	**Y**
Solaris	970	**Y**	**Y**	**Y**	**Y**	*y*
SINIX, DC/OSx						
DEC-OVMS	970	**Y**		**Y**		*y*
Tandem	970	**Y**		**Y**		*y*
Tru64	970 (euc, deckorean)	**Y**	**Y**	**Y**	**Y**	*y*
	1363 (KSC5601)	**Y**	**Y**	**Y**	*y*	**Y**

Notes:
(14) Supported on AS/400 V4R2 or later

Table 114. Conversion support: SIMPLIFIED CHINESE

Product ▼		MVS, OS/400	OS/2, HP-UX, NT	AIX, DEC-OVMS, Tandem, Solaris, Tru64	OS/2,AIX, NT	MVS
	Native CCSID	**935**	**1381**	**1383**	**1386**	**1388**
MVS	935	*y*	**Y**		**Y**	**Y**
	1388	**Y**	**Y**		**Y**	*y*
OS/400	935	*y*	**Y**	**Y**	**Y**	**Y**
OS/2	1381	**Y**	**Y**	**Y**	*y*	**Y**
	1386	**Y**	*y*	**Y**	**Y**	**Y**
AIX	1383 (euc)	**Y**	**Y**	*y*	**Y**	**Y**
	1386 (GBK)	**Y**	**Y**	**Y**	*y*	**Y**
HP-UX	1381 (-15§)	**Y**	*y*			
NCR						
NT	1381##	**Y**	*y*	**Y**	**Y**	**Y**
	1386##	**Y**	**Y**	**Y**	*y*	**Y**
Solaris	1383	**Y**	**Y**	*y*	**Y**	**Y**
SINIX, DC/OSx						
DEC-OVMS	1383	**Y**	**Y**	*y*		
Tandem	1383	**Y**	**Y**	*y*		
Tru64	1383	**Y**	**Y**	*y*	**Y**	**Y**

Notes:

-15§ Is called prc15 and hp15CN on HP-UX

Windows NT uses the code page number 936, but this is best represented by the CCSID of 1386. However not all platforms of MQSeries support this CCSID. On MQSeries for Windows NT 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.

Table 115. Conversion support: TRADITIONAL CHINESE

Product ▼		**MVS, OS/400**	**OS/2, HP-UX**	**OS/2**	**OS/2, AIX, HP-UX, NT, DEC-OVMS, Tandem, Solaris, Tru64**	**AIX, HP-UX, DEC-OVMS, Tandem, Solaris**
	Native CCSID	**937**	**938**	**948**	**950**	**964**
MVS	937	*y*	**Y**	**Y**	**Y**	
OS/400	937	*y*	**Y**	**Y**	**Y**	**Y**
OS/2	938 (PS/55)	**Y**	*y*	**Y**	**Y**	**Y**
	948 (PS/55)	**Y**	**Y**	*y*	**Y**	**Y**
	950 (big5)	**Y**	**Y**	**Y**	*y*	**Y**
AIX	964 (euc)	**Y**	**Y**	**Y**	**Y**	*y*
	950 (big5)	**Y**	**Y**	**Y**	*y*	**Y**
HP-UX	938 (-15§)	**Y**	*y*		**Y**	**Y**
	950 (big5)	**Y**	**Y**		*y*	**Y**
	964 (eucTW)	**Y**	**Y**		**Y**	*y*
NCR						
NT	950	**Y**	**Y**	**Y**	*y*	**Y**
Solaris	964 (euc)	**Y**		**Y**	**Y**	*y*
	950 (big5)	**Y**	**Y**	**Y**	*y*	**Y**
SINIX, DC/OSx						
DEC-OVMS	964 (euc)	**Y**	**Y**	**Y**	**Y**	*y*
	950 (big5)	**Y**	**Y**	**Y**	*y*	**Y**
Tandem	964 (euc)	**Y**	**Y**	**Y**	**Y**	*y*
	950 (big5)	**Y**	**Y**	**Y**	*y*	**Y**
Tru64	950 (big5)	**Y**	**Y**	**Y**	*y*	**Y**

Notes:
-15§ Is called roc15 on HP-UX

OS/390 conversion support

Table 116. MQSeries for OS/390 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 116. MQSeries for OS/390 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
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

OS/390 conversion support

Table 116. MQSeries for OS/390 CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
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
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

Table 116. MQSeries for OS/390 CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
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
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

OS/390 conversion support

Table 116. MQSeries for OS/390 CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
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
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

Table 116. MQSeries for OS/390 CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
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
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

OS/390 conversion support

Table 116. MQSeries for OS/390 CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
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
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

Table 116. MQSeries for OS/390 CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
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
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

OS/390 conversion support

Table 116. MQSeries for OS/390 CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
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
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, 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, 13488, 17584

Table 116. MQSeries for OS/390 CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
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
1162	259, 858-859, 923-924, 1140-1149, 1153-1155, 1160, 5348
1163	924, 1148, 1164, 5354
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
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

OS/390 conversion support

Table 116. MQSeries for OS/390 CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
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
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, 5033, 5046, 9142, 13488, 17584, 25514, 25524, 29620
1371	1200, 1208, 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, 9127, 13223, 13488, 17584
1390	930-932, 939, 942-943, 1200, 1208, 1399, 5026, 5028, 5035, 5038-5039, 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, 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

Table 116. MQSeries for OS/390 CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
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
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

OS/390 conversion support

Table 116. MQSeries for OS/390 CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
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
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

Table 116. MQSeries for OS/390 CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
5484	935-936, 946, 1381, 1386, 1388, 5031, 5477, 5482, 9127, 13223, 25512
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, 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, 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
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

OS/390 conversion support

Table 116. MQSeries for OS/390 CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
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
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

Table 116. MQSeries for OS/390 CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
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
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

OS/390 conversion support

Table 116. MQSeries for OS/390 CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
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
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: MQSeries does not support queue manager Unicode CCSIDs so message header data cannot be encoded in UNICODE.

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 682

MQSeries 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
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	9048
12712	13488	17584	33722		

MQSeries 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	1381	5050
13488	33722				

Note: HP-UX version 10 does not support conversion into or from UTF-8

MQSeries NT, Solaris and Tru64 support for Unicode

On MQSeries for Windows NT, MQSeries for Sun Solaris 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
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	1281
1282	1283	1363	1364	1380	1381
1383	1386	1388	4899	5050	5346
5347	5348	5349	5350	5351	5352
5353	5354	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.

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.

MQSeries for OS/390 support for Unicode

On MQSeries for OS/390 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	1164	1200
1208	1250	1251	1252	1253	1254
1255	1256	1257	1258	1275	1276
1277	1280	1281	1282	1283	1284
1285	1351	1362	1363	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	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.

© Copyright IBM Corp. 1994, 2000

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.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AD/Cycle	AIX	AS/400
BookManager	C/370	CICS
CICS/VSE	DB2	FFST
First Failure Support Technology	IBM	IBMLink
IMS	Integrated Language Environment	Language Environment
MQSeries	MVS	MVS/ESA
OS/2	OS/390	OS/400
Presentation Manager	RACF	S/390
SAA	SP2	System/390
VSE/ESA	VisualAge	

Lotus and Lotus Notes are trademarks of Lotus Development Corporation in the United States, other countries, or both.

Intel is a registered trademark of Intel Corporation in the United States, other countries, or both.

Microsoft, Visual Basic, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java is a registered trademark of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Object attributes

Glossary of terms and abbreviations

This glossary defines MQSeries terms and abbreviations used in this book. If you do not find the term you are looking for, see the Index or the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

This glossary includes terms and definitions from the *American National Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42 Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.

A

abend reason code. A 4-byte hexadecimal code that uniquely identifies a problem with MQSeries for OS/390. A complete list of MQSeries for OS/390 abend reason codes and their explanations is contained in the *MQSeries for OS/390 Messages and Codes* manual.

active log. See *recovery log*.

adapter. An interface between MQSeries for OS/390 and TSO, IMS™, CICS, or batch address spaces. An adapter is an attachment facility that enables applications to access MQSeries services.

address space. The area of virtual storage available for a particular job.

address space identifier (ASID). A unique, system-assigned identifier for an address space.

administrator commands. MQSeries commands used to manage MQSeries objects, such as queues, processes, and namelists.

alert. A message sent to a management services focal point in a network to identify a problem or an impending problem.

alert monitor. In MQSeries for OS/390, a component of the CICS adapter that handles unscheduled events occurring as a result of connection requests to MQSeries for OS/390.

alias queue object. An MQSeries object, the name of which is an alias for a base queue defined to the local queue manager. When an application or a queue manager uses an alias queue, the alias name is resolved and the requested operation is performed on the associated base queue.

allied address space. See *ally*.

ally. An OS/390 address space that is connected to MQSeries for OS/390.

alternate user security. A security feature in which the authority of one user ID can be used by another user ID; for example, to open an MQSeries object.

APAR. Authorized program analysis report.

application environment. The software facilities that are accessible by an application program. On the OS/390 platform, CICS and IMS are examples of application environments.

application log. In Windows NT, a log that records significant application events.

application queue. A queue used by an application.

archive log. See *recovery log*.

ASID. Address space identifier.

asynchronous messaging. A method of communication between programs in which programs place messages on message queues. With asynchronous messaging, the sending program proceeds with its own processing without waiting for a reply to its message. Contrast with *synchronous messaging*.

attribute. One of a set of properties that defines the characteristics of an MQSeries object.

authorization checks. Security checks that are performed when a user tries to issue administration commands against an object, for example to open a queue or connect to a queue manager.

authorization file. In MQSeries on UNIX systems, a file that provides security definitions for an object, a class of objects, or all classes of objects.

authorization service. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a service that provides authority checking of commands and MQI calls for the user identifier associated with the command or call.

authorized program analysis report (APAR). A report of a problem caused by a suspected defect in a current, unaltered release of a program.

B

backout. An operation that reverses all the changes made during the current unit of recovery or unit of

© Copyright IBM Corp. 1994, 2000

work. After the operation is complete, a new unit of recovery or unit of work begins. Contrast with *commit*.

basic mapping support (BMS). An interface between CICS and application programs that formats input and output display data and routes multiple-page output messages without regard for control characters used by various terminals.

BMS. Basic mapping support.

bootstrap data set (BSDS). A VSAM data set that contains:

- An inventory of all active and archived log data sets known to MQSeries for OS/390
- A wrap-around inventory of all recent MQSeries for OS/390 activity

The BSDS is required if the MQSeries for OS/390 subsystem has to be restarted.

browse. In message queuing, to use the MQGET call to copy a message without removing it from the queue. See also *get*.

browse cursor. In message queuing, an indicator used when browsing a queue to identify the message that is next in sequence.

BSDS. Bootstrap data set.

buffer pool. An area of main storage used for MQSeries for OS/390 queues, messages, and object definitions. See also *page set*.

C

call back. In MQSeries, a requester message channel initiates a transfer from a sender channel by first calling the sender, then closing down and awaiting a call back.

CCF. Channel control function.

CCSID. Coded character set identifier.

CDF. Channel definition file.

channel. See *message channel*.

channel control function (CCF). In MQSeries, a program to move messages from a transmission queue to a communication link, and from a communication link to a local queue, together with an operator panel interface to allow the setup and control of channels.

channel definition file (CDF). In MQSeries, a file containing communication channel definitions that associate transmission queues with communication links.

channel event. An event indicating that a channel instance has become available or unavailable. Channel events are generated on the queue managers at both ends of the channel.

checkpoint. A time when significant information is written on the log. Contrast with *syncpoint*. In MQSeries on UNIX systems, the point in time when a data record described in the log is the same as the data record in the queue. Checkpoints are generated automatically and are used during the system restart process.

CI. Control interval.

circular logging. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the process of keeping all restart data in a ring of log files. Logging fills the first file in the ring and then moves on to the next, until all the files are full. At this point, logging goes back to the first file in the ring and starts again, if the space has been freed or is no longer needed. Circular logging is used during restart recovery, using the log to roll back transactions that were in progress when the system stopped. Contrast with *linear logging*.

CL. Control Language.

client. A run-time component that provides access to queuing services on a server for local user applications. The queues used by the applications reside on the server. See also *MQSeries client*.

client application. An application, running on a workstation and linked to a client, that gives the application access to queuing services on a server.

client connection channel type. The type of MQI channel definition associated with an MQSeries client. See also *server connection channel type*.

cluster. A network of queue managers that are logically associated in some way.

coded character set identifier (CCSID). The name of a coded set of characters and their code point assignments.

command. In MQSeries, an administration instruction that can be carried out by the queue manager.

command prefix (CPF). In MQSeries for OS/390, a character string that identifies the queue manager to which MQSeries for OS/390 commands are directed, and from which MQSeries for OS/390 operator messages are received.

command processor. The MQSeries component that processes commands.

command server. The MQSeries component that reads commands from the system-command input queue, verifies them, and passes valid commands to the command processor.

commit. An operation that applies all the changes made during the current unit of recovery or unit of work. After the operation is complete, a new unit of recovery or unit of work begins. Contrast with *backout*.

completion code. A return code indicating how an MQI call has ended.

configuration file. In MQSeries on UNIX systems, MQSeries for AS/400, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a file that contains configuration information related to, for example, logs, communications, or installable services. Synonymous with *.ini file*. See also *stanza*.

connect. To provide a queue manager connection handle, which an application uses on subsequent MQI calls. The connection is made either by the MQCONN or MQCONNX call, or automatically by the MQOPEN call.

connection handle. The identifier or token by which a program accesses the queue manager to which it is connected.

context. Information about the origin of a message.

context security. In MQSeries, a method of allowing security to be handled such that messages are obliged to carry details of their origins in the message descriptor.

control command. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a command that can be entered interactively from the operating system command line. Such a command requires only that the MQSeries product be installed; it does not require a special utility or program to run it.

control interval (CI). A fixed-length area of direct access storage in which VSAM stores records and creates distributed free spaces. The control interval is the unit of information that VSAM transmits to or from direct access storage.

Control Language (CL). In MQSeries for AS/400, a language that can be used to issue commands, either at the command line or by writing a CL program.

controlled shutdown. See *quiesced shutdown*.

CPF. Command prefix.

coupling facility. On OS/390, a special logical partition that provides high-speed caching, list processing, and locking functions in a parallel sysplex.

D

DAE. Dump analysis and elimination.

data conversion interface (DCI). The MQSeries interface to which customer- or vendor-written programs that convert application data between different machine encodings and CCSIDs must conform. A part of the MQSeries Framework.

datagram. The simplest message that MQSeries supports. This type of message does not require a reply.

DCE. Distributed Computing Environment.

DCI. Data conversion interface.

dead-letter queue (DLQ). A queue to which a queue manager or application sends messages that it cannot deliver to their correct destination.

dead-letter queue handler. An MQSeries-supplied utility that monitors a dead-letter queue (DLQ) and processes messages on the queue in accordance with a user-written rules table.

default object. A definition of an object (for example, a queue) with all attributes defined. If a user defines an object but does not specify all possible attributes for that object, the queue manager uses default attributes in place of any that were not specified.

deferred connection. A pending event that is activated when a CICS subsystem tries to connect to MQSeries for OS/390 before MQSeries for OS/390 has been started.

distributed application. In message queuing, a set of application programs that can each be connected to a different queue manager, but that collectively constitute a single application.

Distributed Computing Environment (DCE). Middleware that provides some basic services, making the development of distributed applications easier. DCE is defined by the Open Software Foundation (OSF).

distributed queue management (DQM). In message queuing, the setup and control of message channels to queue managers on other systems.

DLQ. Dead-letter queue.

DQM. Distributed queue management.

dual logging. A method of recording MQSeries for OS/390 activity, where each change is recorded on two data sets, so that if a restart is necessary and one data set is unreadable, the other can be used. Contrast with *single logging*.

dual mode. See *dual logging*.

dump analysis and elimination (DAE). An OS/390 service that enables an installation to suppress SVC dumps and ABEND SYSUDUMP dumps that are not needed because they duplicate previously written dumps.

dynamic queue. A local queue created when a program opens a model queue object. See also *permanent dynamic queue* and *temporary dynamic queue*.

E

environment. See *application environment*.

ESM. External security manager.

ESTAE. Extended specify task abnormal exit.

event. See *channel event*, *instrumentation event*, *performance event*, and *queue manager event*.

event data. In an event message, the part of the message data that contains information about the event (such as the queue manager name, and the application that gave rise to the event). See also *event header*.

event header. In an event message, the part of the message data that identifies the event type of the reason code for the event.

event log. See *application log*.

event message. Contains information (such as the category of event, the name of the application that caused the event, and queue manager statistics) relating to the origin of an instrumentation event in a network of MQSeries systems.

event queue. The queue onto which the queue manager puts an event message after it detects an event. Each category of event (queue manager, performance, or channel event) has its own event queue.

Event Viewer. A tool provided by Windows NT to examine and manage log files.

extended specify task abnormal exit (ESTAE). An OS/390 macro that provides recovery capability and gives control to the specified exit routine for processing, diagnosing an abend, or specifying a retry address.

external security manager (ESM). A security product that is invoked by the OS/390 System Authorization Facility. RACF is an example of an ESM.

F

FFST. First Failure Support Technology™.

FIFO. First-in-first-out.

First Failure Support Technology (FFST). Used by MQSeries on UNIX systems, MQSeries for OS/2 Warp, MQSeries for Windows NT, and MQSeries for AS/400 to detect and report software problems.

first-in-first-out (FIFO). A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time. (A)

forced shutdown. A type of shutdown of the CICS adapter where the adapter immediately disconnects from MQSeries for OS/390, regardless of the state of any currently active tasks. Contrast with *quiesced shutdown*.

Framework. In MQSeries, a collection of programming interfaces that allow customers or vendors to write programs that extend or replace certain functions provided in MQSeries products. The interfaces are:

- MQSeries data conversion interface (DCI)
- MQSeries message channel interface (MCI)
- MQSeries name service interface (NSI)
- MQSeries security enabling interface (SEI)
- MQSeries trigger monitor interface (TMI)

FRR. Functional recovery routine.

functional recovery routine (FRR). An OS/390 recovery/termination manager facility that enables a recovery routine to gain control in the event of a program interrupt.

G

GCPC. Generalized command preprocessor.

generalized command preprocessor (GCPC). An MQSeries for OS/390 component that processes MQSeries commands and runs them.

Generalized Trace Facility (GTF). An OS/390 service program that records significant system events, such as supervisor calls and start I/O operations, for the purpose of problem determination.

get. In message queuing, to use the MQGET call to remove a message from a queue.

global trace. An MQSeries for OS/390 trace option where the trace data comes from the entire MQSeries for OS/390 subsystem.

globally-defined object. On OS/390, an object whose definition is stored in the shared repository. The object is available to all queue managers in the queue-sharing group. See also *locally-defined object*.

GTF. Generalized Trace Facility.

H

handle. See *connection handle* and *object handle*.

hardened message. A message that is written to auxiliary (disk) storage so that the message will not be lost in the event of a system failure. See also *persistent message*.

I

ILE. Integrated Language Environment.

immediate shutdown. In MQSeries, a shutdown of a queue manager that does not wait for applications to disconnect. Current MQI calls are allowed to complete, but new MQI calls fail after an immediate shutdown has been requested. Contrast with *quiesced shutdown* and *preemptive shutdown*.

inbound channel. A channel that receives messages from another queue manager. See also *shared inbound channel*.

in-doubt unit of recovery. In MQSeries, the status of a unit of recovery for which a syncpoint has been requested but not yet confirmed.

Integrated Language Environment® (ILE). The AS/400 Integrated Language Environment. This replaces the AS/400 Original Program Model (OPM).

.ini file. See *configuration file*.

initialization input data sets. Data sets used by MQSeries for OS/390 when it starts up.

initiation queue. A local queue on which the queue manager puts trigger messages.

input/output parameter. A parameter of an MQI call in which you supply information when you make the call, and in which the queue manager changes the information when the call completes or fails.

input parameter. A parameter of an MQI call in which you supply information when you make the call.

installable services. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, additional functionality provided as independent components. The installation of each component is optional: in-house or third-party components can be used instead. See also *authorization service*, *name service*, and *user identifier service*.

instrumentation event. A facility that can be used to monitor the operation of queue managers in a network of MQSeries systems. MQSeries provides instrumentation events for monitoring queue manager resource definitions, performance conditions, and channel conditions. Instrumentation events can be used by a user-written reporting mechanism in an administration application that displays the events to a system operator. They also allow applications acting as agents for other administration networks to monitor reports and create the appropriate alerts.

Interactive Problem Control System (IPCS). A component of OS/390 that permits online problem management, interactive problem diagnosis, online debugging for disk-resident abend dumps, problem tracking, and problem reporting.

Interactive System Productivity Facility (ISPF). An IBM licensed program that serves as a full-screen editor and dialog manager. It is used for writing application programs, and provides a means of generating standard screen panels and interactive dialogues between the application programmer and terminal user.

IPCS. Interactive Problem Control System.

ISPF. Interactive System Productivity Facility.

L

linear logging. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the process of keeping restart data in a sequence of files. New files are added to the sequence as necessary. The space in which the data is written is not reused until the queue manager is restarted. Contrast with *circular logging*.

listener. In MQSeries distributed queuing, a program that monitors for incoming network connections.

local definition. An MQSeries object belonging to a local queue manager.

local definition of a remote queue. An MQSeries object belonging to a local queue manager. This object defines the attributes of a queue that is owned by another queue manager. In addition, it is used for queue-manager aliasing and reply-to-queue aliasing.

local queue. A queue that belongs to the local queue manager. A local queue can contain a list of messages waiting to be processed. Contrast with *remote queue*.

local queue manager. The queue manager to which a program is connected and that provides message queuing services to the program. Queue managers to which a program is not connected are called *remote queue managers*, even if they are running on the same system as the program.

locale. On UNIX systems, a subset of a user's environment that defines conventions for a specific culture (such as time, numeric, or monetary formatting and character classification, collation, or conversion). The queue manager CCSID is derived from the locale of the user ID that created the queue manager.

locally-defined object. On OS/390, an object whose definition is stored on page set zero. The definition can be accessed only by the queue manager that defined it. Also known as a *privately-defined object*.

log. In MQSeries, a file recording the work done by queue managers while they receive, transmit, and deliver messages, to enable them to recover in the event of failure.

log control file. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the file containing information needed to monitor the use of log files (for example, their size and location, and the name of the next available file).

log file. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a file in which all significant changes to the data controlled by a queue manager are recorded. If the primary log files become full, MQSeries allocates secondary log files.

logical unit of work (LUW). See *unit of work*.

M

machine check interrupt. An interruption that occurs as a result of an equipment malfunction or error. A machine check interrupt can be either hardware recoverable, software recoverable, or nonrecoverable.

MCA. Message channel agent.

MCI. Message channel interface.

media image. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the sequence of log records that contain an image of an object. The object can be recreated from this image.

message. In message queuing applications, a communication sent between programs. In system programming, information intended for the terminal operator or system administrator.

message channel. In distributed message queuing, a mechanism for moving messages from one queue manager to another. A message channel comprises two message channel agents (a sender at one end and a receiver at the other end) and a communication link. Contrast with *MQI channel*.

message channel agent (MCA). A program that transmits prepared messages from a transmission queue to a communication link, or from a communication link to a destination queue. See also *message queue interface*.

message channel interface (MCI). The MQSeries interface to which customer- or vendor-written programs that transmit messages between an MQSeries queue manager and another messaging system must conform. A part of the MQSeries Framework.

message descriptor. Control information describing the message format and presentation that is carried as part of an MQSeries message. The format of the message descriptor is defined by the MQMD structure.

message priority. In MQSeries, an attribute of a message that can affect the order in which messages on a queue are retrieved, and whether a trigger event is generated.

message queue. Synonym for *queue*.

message queue interface (MQI). The programming interface provided by the MQSeries queue managers. This programming interface allows application programs to access message queuing services.

message queuing. A programming technique in which each program within an application communicates with the other programs by putting messages on queues.

message sequence numbering. A programming technique in which messages are given unique numbers during transmission over a communication link. This enables the receiving process to check whether all messages are received, to place them in a queue in the original order, and to discard duplicate messages.

messaging. See *synchronous messaging* and *asynchronous messaging*.

model queue object. A set of queue attributes that act as a template when a program creates a dynamic queue.

MQAI. MQSeries Administration Interface.

MQI. Message queue interface.

MQI channel. Connects an MQSeries client to a queue manager on a server system, and transfers only MQI calls and responses in a bidirectional manner. Contrast with *message channel*.

MQSC. MQSeries commands.

MQSeries. A family of IBM licensed programs that provides message queuing services.

MQSeries Administration Interface (MQAI). A programming interface to MQSeries.

MQSeries client. Part of an MQSeries product that can be installed on a system without installing the full queue manager. The MQSeries client accepts MQI calls from applications and communicates with a queue manager on a server system.

MQSeries commands (MQSC). Human readable commands, uniform across all platforms, that are used to manipulate MQSeries objects.

N

namelist. An MQSeries object that contains a list of names, for example, queue names.

name service. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the facility that determines which queue manager owns a specified queue.

name service interface (NSI). The MQSeries interface to which customer- or vendor-written programs that resolve queue-name ownership must conform. A part of the MQSeries Framework.

name transformation. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, an internal process that changes a queue manager name so that it is unique and valid for the system being used. Externally, the queue manager name remains unchanged.

New Technology File System (NTFS). A Windows NT recoverable file system that provides security for files.

nonpersistent message. A message that does not survive a restart of the queue manager. Contrast with *persistent message*.

NSI. Name service interface.

NTFS. New Technology File System.

null character. The character that is represented by X'00'.

O

OAM. Object authority manager.

object. In MQSeries, an object is a queue manager, a queue, a process definition, a channel, a namelist, or a storage class (OS/390 only).

object authority manager (OAM). In MQSeries on UNIX systems, MQSeries for AS/400, and MQSeries for Windows NT, the default authorization service for command and object management. The OAM can be replaced by, or run in combination with, a customer-supplied security service.

object descriptor. A data structure that identifies a particular MQSeries object. Included in the descriptor are the name of the object and the object type.

object handle. The identifier or token by which a program accesses the MQSeries object with which it is working.

off-loading. In MQSeries for OS/390, an automatic process whereby a queue manager's active log is transferred to its archive log.

OPM. Original Program Model.

Original Program Model (OPM). The AS/400 Original Program Model. This is no longer supported on MQSeries. It is replaced by the Integrated Language Environment (ILE).

OTMA. Open Transaction Manager Access.

outbound channel. A channel that takes messages from a transmission queue and sends them to another queue manager. See also *shared outbound channel*.

output log-buffer. In MQSeries for OS/390, a buffer that holds recovery log records before they are written to the archive log.

output parameter. A parameter of an MQI call in which the queue manager returns information when the call completes or fails.

P

page set. A VSAM data set used when MQSeries for OS/390 moves data (for example, queues and messages) from buffers in main storage to permanent backing storage (DASD).

PCF. Programmable command format.

PCF command. See *programmable command format*.

pending event. An unscheduled event that occurs as a result of a connect request from a CICS adapter.

percolation. In error recovery, the passing along a preestablished path of control from a recovery routine to a higher-level recovery routine.

performance event. A category of event indicating that a limit condition has occurred.

performance trace. An MQSeries trace option where the trace data is to be used for performance analysis and tuning.

permanent dynamic queue. A dynamic queue that is deleted when it is closed only if deletion is explicitly requested. Permanent dynamic queues are recovered if the queue manager fails, so they can contain persistent messages. Contrast with *temporary dynamic queue*.

persistent message. A message that survives a restart of the queue manager. Contrast with *nonpersistent message*.

ping. In distributed queuing, a diagnostic aid that uses the exchange of a test message to confirm that a message channel or a TCP/IP connection is functioning.

platform. In MQSeries, the operating system under which a queue manager is running.

point of recovery. In MQSeries for OS/390, the term used to describe a set of backup copies of MQSeries for OS/390 page sets and the corresponding log data sets required to recover these page sets. These backup copies provide a potential restart point in the event of page set loss (for example, page set I/O error).

preemptive shutdown. In MQSeries, a shutdown of a queue manager that does not wait for connected applications to disconnect, nor for current MQI calls to complete. Contrast with *immediate shutdown* and *quiesced shutdown*.

principal. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a term used for a user identifier. Used by the object authority manager for checking authorizations to system resources.

privately-defined object. In OS/390, an object whose definition is stored on page set zero. The definition can be accessed only by the queue manager that defined it. Also known as a *locally-defined object*.

process definition object. An MQSeries object that contains the definition of an MQSeries application. For example, a queue manager uses the definition when it works with trigger messages.

programmable command format (PCF). A type of MQSeries message used by:

- User administration applications, to put PCF commands onto the system command input queue of a specified queue manager
- User administration applications, to get the results of a PCF command from a specified queue manager
- A queue manager, as a notification that an event has occurred

Contrast with *MQSC*.

program temporary fix (PTF). A solution or by-pass of a problem diagnosed by IBM field engineering as the result of a defect in a current, unaltered release of a program.

PTF. Program temporary fix.

Q

queue. An MQSeries object. Message queuing applications can put messages on, and get messages from, a queue. A queue is owned and maintained by a queue manager. Local queues can contain a list of messages waiting to be processed. Queues of other types cannot contain messages—they point to other queues, or can be used as models for dynamic queues.

queue manager. A system program that provides queuing services to applications. It provides an application programming interface so that programs can access messages on the queues that the queue manager owns. See also *local queue manager* and *remote queue manager*. An MQSeries object that defines the attributes of a particular queue manager.

queue manager event. An event that indicates:

- An error condition has occurred in relation to the resources used by a queue manager. For example, a queue is unavailable.
- A significant change has occurred in the queue manager. For example, a queue manager has stopped or started.

queue-sharing group. In MQSeries for OS/390, a group of queue managers in the same sysplex that can access a single set of object definitions stored in the shared repository, and a single set of shared queues stored in the coupling facility. See also *shared queue*.

queuing. See *message queuing*.

quiesced shutdown. In MQSeries, a shutdown of a queue manager that allows all connected applications to disconnect. Contrast with *immediate shutdown* and *preemptive shutdown*. A type of shutdown of the CICS adapter where the adapter disconnects from MQSeries, but only after all the currently active tasks have been completed. Contrast with *forced shutdown*.

quiescing. In MQSeries, the state of a queue manager prior to it being stopped. In this state, programs are allowed to finish processing, but no new programs are allowed to start.

R

RBA. Relative byte address.

reason code. A return code that describes the reason for the failure or partial success of an MQI call.

receiver channel. In message queuing, a channel that responds to a sender channel, takes messages from a communication link, and puts them on a local queue.

recovery log. In MQSeries for OS/390, data sets containing information needed to recover messages, queues, and the MQSeries subsystem. MQSeries for OS/390 writes each record to a data set called the *active log*. When the active log is full, its contents are off-loaded to a DASD or tape data set called the *archive log*. Synonymous with *log*.

recovery termination manager (RTM). A program that handles all normal and abnormal termination of tasks by passing control to a recovery routine associated with the terminating function.

Registry. In Windows NT, a secure database that provides a single source for system and application configuration data.

Registry Editor. In Windows NT, the program item that allows the user to edit the Registry.

Registry Hive. In Windows NT, the structure of the data stored in the Registry.

relative byte address (RBA). The displacement in bytes of a stored record or control interval from the beginning of the storage space allocated to the data set to which it belongs.

remote queue. A queue belonging to a remote queue manager. Programs can put messages on remote queues, but they cannot get messages from remote queues. Contrast with *local queue*.

remote queue manager. To a program, a queue manager that is not the one to which the program is connected.

remote queue object. See *local definition of a remote queue*.

remote queuing. In message queuing, the provision of services to enable applications to put messages on queues belonging to other queue managers.

reply message. A type of message used for replies to request messages. Contrast with *request message* and *report message*.

reply-to queue. The name of a queue to which the program that issued an MQPUT call wants a reply message or report message sent.

report message. A type of message that gives information about another message. A report message can indicate that a message has been delivered, has arrived at its destination, has expired, or could not be processed for some reason. Contrast with *reply message* and *request message*.

requester channel. In message queuing, a channel that may be started remotely by a sender channel. The requester channel accepts messages from the sender channel over a communication link and puts the messages on the local queue designated in the message. See also *server channel*.

request message. A type of message used to request a reply from another program. Contrast with *reply message* and *report message*.

RESLEVEL. In MQSeries for OS/390, an option that controls the number of CICS user IDs checked for API-resource security in MQSeries for OS/390.

resolution path. The set of queues that are opened when an application specifies an alias or a remote queue on input to an MQOPEN call.

resource. Any facility of the computing system or operating system required by a job or task. In MQSeries for OS/390, examples of resources are buffer pools, page sets, log data sets, queues, and messages.

resource manager. An application, program, or transaction that manages and controls access to shared resources such as memory buffers and data sets. MQSeries, CICS, and IMS are resource managers.

Resource Recovery Services (RRS). An OS/390 facility that provides 2-phase syncpoint support across participating resource managers.

responder. In distributed queuing, a program that replies to network connection requests from another system.

resynch. In MQSeries, an option to direct a channel to start up and resolve any in-doubt status messages, but without restarting message transfer.

return codes. The collective name for completion codes and reason codes.

rollback. Synonym for *back out*.

RRS. Resource Recovery Services.

RTM. Recovery termination manager.

rules table. A control file containing one or more rules that the dead-letter queue handler applies to messages on the DLQ.

S

SAF. System Authorization Facility.

SDWA. System diagnostic work area.

security enabling interface (SEI). The MQSeries interface to which customer- or vendor-written programs that check authorization, supply a user identifier, or perform authentication must conform. A part of the MQSeries Framework.

SEI. Security enabling interface.

sender channel. In message queuing, a channel that initiates transfers, removes messages from a transmission queue, and moves them over a communication link to a receiver or requester channel.

sequential delivery. In MQSeries, a method of transmitting messages with a sequence number so that the receiving channel can reestablish the message sequence when storing the messages. This is required where messages must be delivered only once, and in the correct order.

sequential number wrap value. In MQSeries, a method of ensuring that both ends of a communication link reset their current message sequence numbers at the same time. Transmitting messages with a sequence

number ensures that the receiving channel can reestablish the message sequence when storing the messages.

server. (1) In MQSeries, a queue manager that provides queue services to client applications running on a remote workstation. (2) The program that responds to requests for information in the particular two-program, information-flow model of client/server. See also *client*.

server channel. In message queuing, a channel that responds to a requester channel, removes messages from a transmission queue, and moves them over a communication link to the requester channel.

server connection channel type. The type of MQI channel definition associated with the server that runs a queue manager. See also *client connection channel type*.

service interval. A time interval, against which the elapsed time between a put or a get and a subsequent get is compared by the queue manager in deciding whether the conditions for a service interval event have been met. The service interval for a queue is specified by a queue attribute.

service interval event. An event related to the service interval.

session ID. In MQSeries for OS/390, the CICS-unique identifier that defines the communication link to be used by a message channel agent when moving messages from a transmission queue to a link.

shared inbound channel. In MQSeries for OS/390, a channel that was started by a listener using the group port. The channel definition of a shared channel can be stored either on page set zero (private) or in the shared repository (global).

shared outbound channel. In MQSeries for OS/390, a channel that moves messages from a shared transmission queue. The channel definition of a shared channel can be stored either on page set zero (private) or in the shared repository (global).

shared queue. In MQSeries for OS/390, a type of local queue. The messages on the queue are stored in the *coupling facility* and can be accessed by one or more queue managers in a *queue-sharing group*. The definition of the queue is stored in the *shared repository*.

shared repository. In MQSeries for OS/390, a shared DB2 database that is used to hold object definitions that have been defined globally.

shutdown. See *immediate shutdown*, *preemptive shutdown*, and *quiesced shutdown*.

signaling. In MQSeries for OS/390 and MQSeries for Windows 2.1, a feature that allows the operating system to notify a program when an expected message arrives on a queue.

single logging. A method of recording MQSeries for OS/390 activity where each change is recorded on one data set only. Contrast with *dual logging*.

single-phase backout. A method in which an action in progress must not be allowed to finish, and all changes that are part of that action must be undone.

single-phase commit. A method in which a program can commit updates to a queue without coordinating those updates with updates the program has made to resources controlled by another resource manager. Contrast with *two-phase commit*.

SIT. System initialization table.

stanza. A group of lines in a configuration file that assigns a value to a parameter modifying the behavior of a queue manager, client, or channel. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a configuration (.ini) file may contain a number of stanzas.

storage class. In MQSeries for OS/390, a storage class defines the page set that is to hold the messages for a particular queue. The storage class is specified when the queue is defined.

store and forward. The temporary storing of packets, messages, or frames in a data network before they are retransmitted toward their destination.

subsystem. In OS/390, a group of modules that provides function that is dependent on OS/390. For example, MQSeries for OS/390 is an OS/390 subsystem.

supervisor call (SVC). An OS/390 instruction that interrupts a running program and passes control to the supervisor so that it can perform the specific service indicated by the instruction.

SVC. Supervisor call.

switch profile. In MQSeries for OS/390, a RACF profile used when MQSeries starts up or when a refresh security command is issued. Each switch profile that MQSeries detects turns off checking for the specified resource.

symptom string. Diagnostic information displayed in a structured format designed for searching the IBM software support database.

synchronous messaging. A method of communication between programs in which programs place messages on message queues. With synchronous messaging, the

sending program waits for a reply to its message before resuming its own processing. Contrast with *asynchronous messaging*.

syncpoint. An intermediate or end point during processing of a transaction at which the transaction's protected resources are consistent. At a syncpoint, changes to the resources can safely be committed, or they can be backed out to the previous syncpoint.

System Authorization Facility (SAF). An OS/390 facility through which MQSeries for OS/390 communicates with an external security manager such as RACF.

system.command.input queue. A local queue on which application programs can put MQSeries commands. The commands are retrieved from the queue by the command server, which validates them and passes them to the command processor to be run.

system control commands. Commands used to manipulate platform-specific entities such as buffer pools, storage classes, and page sets.

system diagnostic work area (SDWA). Data recorded in a SYS1.LOGREC entry, which describes a program or hardware error.

system initialization table (SIT). A table containing parameters used by CICS on start up.

SYS1.LOGREC. A service aid containing information about program and hardware errors.

T

TACL. Tandem Advanced Command Language.

target library high-level qualifier (thlqual). High-level qualifier for OS/390 target data set names.

task control block (TCB). An OS/390 control block used to communicate information about tasks within an address space that are connected to an OS/390 subsystem such as MQSeries for OS/390 or CICS.

task switching. The overlapping of I/O operations and processing between several tasks. In MQSeries for OS/390, the task switcher optimizes performance by allowing some MQI calls to be executed under subtasks rather than under the main CICS TCB.

TCB. Task control block.

temporary dynamic queue. A dynamic queue that is deleted when it is closed. Temporary dynamic queues are not recovered if the queue manager fails, so they can contain nonpersistent messages only. Contrast with *permanent dynamic queue*.

teraspace. In MQSeries for AS/400, a form of shared memory introduced in OS/400 V4R4.

termination notification. A pending event that is activated when a CICS subsystem successfully connects to MQSeries for OS/390.

thlqual. Target library high-level qualifier.

thread. In MQSeries, the lowest level of parallel execution available on an operating system platform.

time-independent messaging. See *asynchronous messaging*.

TMI. Trigger monitor interface.

trace. In MQSeries, a facility for recording MQSeries activity. The destinations for trace entries can include GTF and the system management facility (SMF).

tranid. See *transaction identifier*.

transaction identifier. In CICS, a name that is specified when the transaction is defined, and that is used to invoke the transaction.

transmission program. See *message channel agent*.

transmission queue. A local queue on which prepared messages destined for a remote queue manager are temporarily stored.

trigger event. An event (such as a message arriving on a queue) that causes a queue manager to create a trigger message on an initiation queue.

triggering. In MQSeries, a facility allowing a queue manager to start an application automatically when predetermined conditions on a queue are satisfied.

trigger message. A message containing information about the program that a trigger monitor is to start.

trigger monitor. A continuously-running application serving one or more initiation queues. When a trigger message arrives on an initiation queue, the trigger monitor retrieves the message. It uses the information in the trigger message to start a process that serves the queue on which a trigger event occurred.

trigger monitor interface (TMI). The MQSeries interface to which customer- or vendor-written trigger monitor programs must conform. A part of the MQSeries Framework.

two-phase commit. A protocol for the coordination of changes to recoverable resources when more than one resource manager is used by a single transaction. Contrast with *single-phase commit*.

U

UIS. User identifier service.

undelivered-message queue. See *dead-letter queue*.

undo/redo record. A log record used in recovery. The redo part of the record describes a change to be made to an MQSeries object. The undo part describes how to back out the change if the work is not committed.

unit of recovery. A recoverable sequence of operations within a single resource manager. Contrast with *unit of work*.

unit of work. A recoverable sequence of operations performed by an application between two points of consistency. A unit of work begins when a transaction starts or after a user-requested syncpoint. It ends either at a user-requested syncpoint or at the end of a transaction. Contrast with *unit of recovery*.

user identifier service (UIS). In MQSeries for OS/2 Warp, the facility that allows MQI applications to associate a user ID, other than the default user ID, with MQSeries messages.

utility. In MQSeries, a supplied set of programs that provide the system operator or system administrator with facilities in addition to those provided by the MQSeries commands. Some utilities invoke more than one function.

Bibliography

This section describes the documentation available for all current MQSeries products.

MQSeries cross-platform publications

Most of these publications, which are sometimes referred to as the MQSeries "family" books, apply to all MQSeries Level 2 products. The latest MQSeries Level 2 products are:

- MQSeries for AIX, V5.1
- MQSeries for AS/400, V5.1
- MQSeries for AT&T GIS UNIX V2.2
- MQSeries for Compaq (DIGITAL) OpenVMS, V2.2.1.1
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for HP-UX, V5.1
- MQSeries for OS/2 Warp, V5.1
- MQSeries for OS/390, V2.2
- MQSeries for SINIX and DC/OSx, V2.2
- MQSeries for Sun Solaris, V5.1
- MQSeries for Sun Solaris, Intel Platform Edition, V5.1
- MQSeries for Tandem NonStop Kernel, V2.2.0.1
- MQSeries for VSE/ESA V2.1
- MQSeries for Windows V2.0
- MQSeries for Windows V2.1
- MQSeries for Windows NT, V5.1

The MQSeries cross-platform publications are:

- *MQSeries Brochure*, G511-1908
- *An Introduction to Messaging and Queuing*, GC33-0805
- *MQSeries Intercommunication*, SC33-1872
- *MQSeries Queue Manager Clusters*, SC34-5349
- *MQSeries Clients*, GC33-1632
- *MQSeries System Administration*, SC33-1873
- *MQSeries MQSC Command Reference*, SC33-1369
- *MQSeries Event Monitoring*, SC34-5760
- *MQSeries Programmable System Management*, SC33-1482
- *MQSeries Administration Interface Programming Guide and Reference*, SC34-5390
- *MQSeries Messages*, GC33-1876
- *MQSeries Application Programming Guide*, SC33-0807
- *MQSeries Application Programming Reference*, SC33-1673
- *MQSeries Programming Interfaces Reference Summary*, SX33-6095
- *MQSeries Using C++*, SC33-1877
- *MQSeries Using Java*, SC34-5456
- *MQSeries Application Messaging Interface*, SC34-5604

MQSeries platform-specific publications

Each MQSeries product is documented in at least one platform-specific publication, in addition to the MQSeries family books.

MQSeries for AIX, V5.1

MQSeries for AIX Quick Beginnings, GC33-1867

MQSeries for AS/400, V5.1

MQSeries for AS/400 Quick Beginnings, GC34-5557

MQSeries for AS/400 System Administration, SC34-5558

MQSeries for AS/400 Application Programming Reference (ILE RPG), SC34-5559

MQSeries for AT&T GIS UNIX V2.2

MQSeries for AT&T GIS UNIX System Management Guide, SC33-1642

MQSeries for Compaq (DIGITAL) OpenVMS, V2.2.1.1

MQSeries for Digital OpenVMS System Management Guide, GC33-1791

MQSeries for Compaq Tru64 UNIX, V5.1

MQSeries for Compaq Tru64 UNIX Quick Beginnings, GC34-5684

MQSeries for HP-UX, V5.1

MQSeries for HP-UX Quick Beginnings, GC33-1869

MQSeries for OS/2 Warp, V5.1

MQSeries for OS/2 Warp Quick Beginnings, GC33-1868

© Copyright IBM Corp. 1994, 2000

MQSeries for OS/390, V2.2

MQSeries for OS/390 Concepts and Planning Guide, GC34-5650

MQSeries for OS/390 System Setup Guide, SC34-5651

MQSeries for OS/390 System Administration Guide, SC34-5652

MQSeries for OS/390 Problem Determination Guide, GC34-5376

MQSeries for OS/390 Messages and Codes, GC34-5375

MQSeries for OS/390 Licensed Program Specifications, GC34-5377

MQSeries for OS/390 Program Directory

MQSeries link for R/3 Version 1.2

MQSeries link for R/3 User's Guide, GC33-1934

MQSeries for SINIX and DC/OSx, V2.2

MQSeries for SINIX and DC/OSx System Management Guide, GC33-1768

MQSeries for Sun Solaris, V5.1

MQSeries for Sun Solaris Quick Beginnings, GC33-1870

MQSeries for Sun Solaris, Intel Platform Edition, V5.1

MQSeries for Sun Solaris, Intel Platform Edition Quick Beginnings, GC34-5851

MQSeries for Tandem NonStop Kernel, V2.2.0.1

MQSeries for Tandem NonStop Kernel System Management Guide, GC33-1893

MQSeries for VSE/ESA V2.1

MQSeries for VSE/ESA Version 2 Release 1 Licensed Program Specifications, GC34-5365

MQSeries for VSE/ESA System Management Guide, GC34-5364

MQSeries for Windows V2.0

MQSeries for Windows User's Guide, GC33-1822

MQSeries for Windows V2.1

MQSeries for Windows User's Guide, GC33-1965

MQSeries for Windows NT, V5.1

MQSeries for Windows NT Quick Beginnings, GC34-5389

MQSeries for Windows NT Using the Component Object Model Interface, SC34-5387

MQSeries LotusScript Extension, SC34-5404

Softcopy books

Most of the MQSeries books are supplied in both hardcopy and softcopy formats.

HTML format

Relevant MQSeries documentation is provided in HTML format with these MQSeries products:

- MQSeries for AIX, V5.1
- MQSeries for AS/400, V5.1
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for HP-UX, V5.1
- MQSeries for OS/2 Warp, V5.1
- MQSeries for OS/390, V2.2
- MQSeries for Sun Solaris, V5.1
- MQSeries for Windows NT, V5.1 (compiled HTML)
- MQSeries link for R/3 V1.2

The MQSeries books are also available in HTML format from the MQSeries product family Web site at:

```
http://www.ibm.com/software/mqseries/
```

Portable Document Format (PDF)

PDF files can be viewed and printed using the Adobe Acrobat Reader.

If you need to obtain the Adobe Acrobat Reader, or would like up-to-date information about the platforms on which the Acrobat Reader is supported, visit the Adobe Systems Inc. Web site at:

```
http://www.adobe.com/
```

PDF versions of relevant MQSeries books are supplied with these MQSeries products:

- MQSeries for AIX, V5.1
- MQSeries for AS/400, V5.1
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for HP-UX, V5.1
- MQSeries for OS/2 Warp, V5.1
- MQSeries for OS/390, V2.2
- MQSeries for Sun Solaris, V5.1
- MQSeries for Windows NT, V5.1
- MQSeries link for R/3 V1.2

PDF versions of all current MQSeries books are also available from the MQSeries product family Web site at:

```
http://www.ibm.com/software/mqseries/
```

BookManager® format

The MQSeries library is supplied in IBM BookManager format on a variety of online library collection kits, including the *Transaction Processing and Data* collection kit, SK2T-0730. You can view the softcopy books in IBM BookManager format using the following IBM licensed programs:

BookManager READ/2
BookManager READ/6000
BookManager READ/DOS
BookManager READ/MVS
BookManager READ/VM
BookManager READ for Windows

PostScript format

The MQSeries library is provided in PostScript (.PS) format with many MQSeries Version 2 products. Books in PostScript format can be printed on a PostScript printer or viewed with a suitable viewer.

Windows Help format

The *MQSeries for Windows User's Guide* is provided in Windows Help format with MQSeries for Windows Version 2.0 and MQSeries for Windows Version 2.1.

MQSeries information available on the Internet

The MQSeries product family Web site is at:

```
http://www.ibm.com/software/mqseries/
```

By following links from this Web site you can:

- Obtain latest information about the MQSeries product family.
- Access the MQSeries books in HTML and PDF formats.
- Download MQSeries SupportPacs.

Index

A

AbendCode field 35
AccountingToken field
  MQMD structure 127
  MQPMR structure 234
ADSDescriptor field 35
aliasing
  queue manager 427
  reply queue 427
AlterationDate attribute
  namelist 459
  process definition 463
  queue 430
  queue manager 470
AlterationTime attribute
  namelist 459
  process definition 463
  queue 430
  queue manager 470
AlternateSecurityId field 194
AlternateUserId field 195
ApplId
  attribute 464
  field
    MQTM structure 267
    MQTMC2 structure 274
ApplIdentityData field 129
ApplOriginData field 130
ApplType
  attribute 464
  field
    MQTM structure 267
    MQTMC2 structure 274
AppOptions field 608
Arabic language support 653
assembler programming language
  notational conventions 26
AttentionId field 36
attributes
  namelist 459
  process definition 463
  queue 427
  queue manager 469
Authenticator field 36, 118
AuthorityEvent attribute 471

B

BackoutCount field 131
BackoutRequeueQName attribute 431
BackoutThreshold attribute 431
BaseQName attribute 431
begin options structure 29
BeginOptions parameter 315
bibliography 703
BookManager 705
Buffer parameter
  declaring 307
  MQGET call 350
  MQPUT call 395
  MQPUT1 call 406
BufferLength parameter
  MQGET call 350
  MQPUT call 394
  MQPUT1 call 406
built-in formats 140

C

C programming language
  data types 17
  functions 16
  header files 16
  initial values for dynamic structures 18
  initial values for structures 18
  manipulating binary strings 17
  manipulating character strings 17
  notational conventions 19
  parameters with undefined data types 17
  use from C++ 19
  using calls 307
calls
  conventions used 305
  detailed description
    MQ_DATA_CONV_EXIT 620
    MQBACK 309
    MQBEGIN 315
    MQCLOSE 319
    MQCMIT 327
    MQCONN 333
    MQCONNX 341
    MQDISC 345
    MQGET 349
    MQINQ 363
    MQOPEN 375
    MQPUT 393
    MQPUT1 405
    MQSET 415
    MQSYNC 423
    MQXCNVC 613
CancelCode field 36
CCSID language support tables 629
CFStrucName attribute 432
ChannelAutoDef attribute 471
ChannelAutoDefEvent attribute 471
ChannelAutoDefExit attribute 472
CharAttrLength parameter
  MQINQ call 368
  MQSET call 417
CharAttrs parameter
  MQINQ call 368
  MQSET call 417
Chinese language support 664, 665
ClientConnOffset field 52
ClientConnPtr field 52
ClusterName attribute 432
ClusterNamelist attribute 433
ClusterWorkloadData attribute 472
ClusterWorkloadExit attribute 473
ClusterWorkloadLength attribute 473
CMQB.BAS Visual Basic header file 26
COBOL programming language
  COPY files 19
  named constants 21
  notational conventions 22
  pointer data type 21
  structures 20
code-page conversions 629
coded character set identifier 473
CodedCharSetId
  attribute 473
  field
    MQCIH structure 37
    MQDH structure 62
    MQDLH structure 71
    MQDXP structure 608
    MQIIH structure 118
    MQMD structure 131
    MQMDE structure 188
    MQRFH structure 237
    MQRFH2 structure 244
    MQRMH structure 253
    MQWIH structure 280
CommandInputQName attribute 474
CommandLevel attribute 474
CommitAbort parameter 423
CommitMode field 118
compatibility mode 339
CompCode field
  MQCIH structure 37
  MQDXP structure 608
  MQRR structure 263
CompCode parameter
  MQBACK call 309
  MQBEGIN call 315
  MQCLOSE call 321
  MQCMIT call 327
  MQCONN call 336
  MQCONNX call 341
  MQDISC call 345
  MQGET call 351
  MQINQ call 368
  MQOPEN call 382
  MQPUT call 395
  MQPUT1 call 406
  MQSET call 417
  MQSYNC call 423
  MQXCNVC call 618
completion code 489
connect options structure 51
ConnectOpts parameter 341
ConnTag field 54
constants, values of 547
  accounting token (MQACT_*) 548
  accounting token type (MQACTT_*) 548
  application type (MQAT_*) 549
  backout hardening (MQQA_*) 572
  begin options (MQBO_*) 549
  begin options structure identifier (MQBO_*) 550
  begin options version (MQBO_*) 550

© Copyright IBM Corp. 1994, 2000

constants, values of 547 *(continued)*
binding (MQBND_*) 549
character attribute selectors (MQCA_*) 550
CICS bridge return code (MQCRC_*) 555
CICS function name (MQCFUNC_*) 552
CICS header ADS descriptor (MQCADSD_*) 551
CICS header conversational task (MQCCT_*) 551
CICS header facility (MQCFAC_*) 552
CICS header flags (MQCIH_*) 553
CICS header get-wait interval (MQCGWI_*) 552
CICS header length (MQCIH_*) 553
CICS header link type (MQCLT_*) 553
CICS header output data length (MQCODL_*) 555
CICS header structure identifier (MQCIH_*) 553
CICS header task end status (MQCTES_*) 555
CICS header transaction start code (MQCSC_*) 555
CICS header unit-of-work control (MQCUOWC_*) 556
CICS header version (MQCIH_*) 553
close options (MQCO_*) 554
coded character set identifier (MQCCSI_*) 551
command level (MQCMDL_*) 553
completion codes (MQCC_*) 551
connect options (MQCNO_*) 554
connect options structure identifier (MQCNO_*) 554
connect options version (MQCNO_*) 554
connection handle (MQHC_*) 562
connection tag (MQCT_*) 555
convert-characters masks and factors (MQDCC_*) 556
convert-characters options (MQDCC_*) 556
correlation identifier (MQCI_*) 552
data-conversion-exit parameter structure identifier (MQDXP_*) 557
data-conversion-exit parameter structure version (MQDXP_*) 558
data-conversion-exit response (MQXDR_*) 584
dead-letter header structure identifier (MQDLH_*) 557
dead-letter header version (MQDLH_*) 557
distribution header flags (MQDHF_*) 557
distribution header structure identifier (MQDH_*) 556
distribution header version (MQDH_*) 557
distribution list support (MQDL_*) 557
encoding (MQENC_*) 558

constants, values of 547 *(continued)*
encoding for binary integers (MQENC_*) 559
encoding for floating-point numbers (MQENC_*) 559
encoding for packed-decimal integers (MQENC_*) 559
encoding masks (MQENC_*) 558
event reporting (MQEVR_*) 559
event reporting (MQQSIE_*) 573
exit command identifier (MQXC_*) 584
exit identifier (MQXT_*) 585
exit parameter block structure identifier (MQXP_*) 584
exit parameter block version (MQXP_*) 584
exit reason (MQXR_*) 585
exit response (MQXCC_*) 584
exit user area (MQXUA_*) 585
expiry interval (MQEI_*) 558
feedback (MQFB_*) 559
format (MQFMT_*) 560
get message options (MQGMO_*) 561
get message options structure identifier (MQGMO_*) 562
get message options version (MQGMO_*) 562
group identifier (MQGI_*) 561
group status (MQGS_*) 562
IMS authenticator (MQIAUT_*) 564
IMS commit mode (MQICM_*) 564
IMS header flags (MQIIH_*) 565
IMS header length (MQIIH_*) 565
IMS header structure identifier (MQIIH_*) 565
IMS header version (MQIIH_*) 565
IMS security scope (MQISS_*) 565
IMS transaction instance identifier (MQITII_*) 566
IMS transaction state (MQITS_*) 566
Index type (MQIT_*) 565
inhibit get (MQQA_*) 572
inhibit put (MQQA_*) 572
integer attribute selectors (MQIA_*) 563
integer attribute value (MQIAV_*) 564
intra-group queuing (MQIGQ_*) 564
intra-group queuing put authority (MQIGQPA_*) 564
lengths of character string and byte fields (MQ_*) 547
match options (MQMO_*) 568
message delivery sequence (MQMDS_*) 567
message descriptor extension flags (MQMDEF_*) 567
message descriptor extension length (MQMDE_*) 566
message descriptor extension structure identifier (MQMDE_*) 567
message descriptor extension version (MQMDE_*) 567
message descriptor structure identifier (MQMD_*) 566

constants, values of 547 *(continued)*
message descriptor version (MQMD_*) 566
message flags (MQMF_*) 567
message-flags masks (MQMF_*) 567
message identifier (MQMI_*) 568
message token (MQMTOK_*) 568
message type (MQMT_*) 568
name count (MQNC_*) 569
object descriptor length (MQOD_*) 569
object descriptor structure identifier (MQOD_*) 569
object descriptor version (MQOD_*) 569
object handle (MQHO_*) 562
object instance identifier (MQOII_*) 569
object type (MQOT_*) 570
open options (MQOO_*) 570
original length (MQOL_*) 569
persistence (MQPER_*) 570
platform (MQPL_*) 570
priority (MQPRI_*) 572
put message options (MQPMO_*) 571
put message options length (MQPMO_*) 571
put message options structure identifier (MQPMO_*) 571
put message options version (MQPMO_*) 571
put message record field flags (MQPMRF_*) 571
queue definition type (MQQDT_*) 572
queue shareability (MQQA_*) 572
queue-sharing group disposition (MQQSGD_*) 573
queue type (MQQT_*) 573
reason codes (MQRC_*) 573
reference message header flags (MQRMHF_*) 580
reference message header structure identifier (MQRMH_*) 579
reference message header version (MQRMH_*) 580
report options (MQRO_*) 580
report-options masks (MQRO_*) 580
returned length (MQRL_*) 579
rules and formatting header flags (MQRFH_*) 579
rules and formatting header length (MQRFH_*) 579
rules and formatting header structure identifier (MQRFH_*) 579
rules and formatting header version (MQRFH_*) 579
scope (MQSCO_*) 580
security identifier (MQSID_*) 581
security identifier type (MQSIDT_*) 581
segment status (MQSS_*) 581
segmentation (MQSEG_*) 581
signal event-control-block completion codes (MQEC_*) 558
syncpoint (MQSP_*) 581

constants, values of 547 *(continued)*
transmission queue header structure identifier 585
transmission queue header version (MQXQH_*) 585
trigger controls (MQTC_*) 581
trigger message (character format) structure identifier (MQTMC_*) 582
trigger message (character format) version (MQTMC_*) 582
trigger message structure identifier (MQTM_*) 582
trigger message version (MQTM_*) 582
trigger type (MQTT_*) 582
undelivered-message header structure identifier (MQDLH_*) 557
undelivered-message header version (MQDLH_*) 557
usage (MQUS_*) 583
wait interval (MQWI_*) 583
workload information header flags (MQWIH_*) 583
workload information header structure identifier (MQWIH_*) 583
workload information header structure length (MQWIH_*) 583
workload information header version (MQWIH_*) 583
Context field 212
ConversationalTask field 37
conversion of report messages 605
conversions, code-page 629
COPY files – COBOL programming language 19
CorrelId field
MQMD structure 132
MQPMR structure 234
CreationDate attribute 433
CreationTime attribute 433
CurrentQDepth attribute 434
CursorPosition field 37
Cyrillic support 646

D

Danish language support 635
data conversion
processing conventions 601
report messages 605
data types, conventions used 15
data types, detailed description
elementary
assembler language 11
C programming language 9
COBOL programming language 10
MQBYTE 7
MQBYTEn 7
MQCHAR 8
MQCHARn 8
MQHCONN 8
MQHOBJ 8
MQLONG 9
MQPTR 9
overview 7
PL/I language 10
data types, detailed description *(continued)*
elementary *(continued)*
TAL programming language 12
Visual Basic 12
structure
MQBO 29
MQCIH 33
MQCNO 51
MQDH 61
MQDLH 69
MQDXP 607
MQGMO 81
MQIIH 117
MQMD 125
MQMDE 185
MQOD 193
MQOR 209
MQPMO 211
MQPMR 233
MQRFH 237
MQRFH2 243
MQRMH 251
MQRR 263
MQTM 265
MQTMC2 273
MQWIH 279
MQXP 285
MQXQH 291
programming considerations 14
rules 14
DataConvExitParms parameter 620
DataLength
field, MQDXP structure 608
parameter
MQGET call 351
MQXCNVC call 617
DataLogicalLength field 253
DataLogicalOffset field 253
DataLogicalOffset2 field 254
dead-letter header structure 69
DeadLetterQName attribute 476
DefBind attribute 434
DefinitionType attribute 435
DefInputOpenOption attribute 436
DefPersistence attribute 436
DefPriority attribute 437
DefXmitQName attribute 477
DestEnvLength field 254
DestEnvOffset field 254
DestNameLength field 254
DestNameOffset field 255
DestQMgrName field 71
DestQName field 72
DistLists attribute 438, 477
distribution header structure 61
distribution lists 438, 477
dynamic queue 375
DynamicQName field 196

E

Eastern European languages support 645
Encoding field
MQCIH structure 37
MQDH structure 63
MQDLH structure 72
Encoding field *(continued)*
MQDXP structure 609
MQIIH structure 119
MQMD structure 133
MQMDE structure 188
MQRFH structure 238
MQRFH2 structure 244
MQRMH structure 255
MQWIH structure 280
using 589
EnvData
attribute 465
field
MQTM structure 268
MQTMC2 structure 274
environment variable – MQ_CONNECT_TYPE 56
ErrorOffset field 37
Estonian language support 647
exit parameter block 285
ExitCommand field 285
ExitId field 286
ExitOptions field 609
ExitParmCount field 286
ExitReason field 286
ExitResponse field
MQDXP structure 609
MQXP structure 287
ExitUserArea field 287
Expiry field 134

F

Facility field 38
FacilityKeepTime field 38
FacilityLike field 38
Farsi support 654
Feedback field
MQMD structure 136
MQPMR structure 234
Finnish language support 637
Flags field
MQCIH structure 38
MQDH structure 63
MQIIH structure 119
MQMDE structure 188
MQRFH structure 238
MQRFH2 structure 244
MQRMH structure 255
MQWIH structure 280
fonts in this book xxii
form files (.BAS) 26
Format field
MQCIH structure 39
MQDH structure 64
MQDLH structure 72
MQIIH structure 119
MQMD structure 140
MQMDE structure 188
MQRFH structure 238
MQRFH2 structure 244
MQRMH structure 255
MQWIH structure 280
formats built-in 140
French language support 641
Function field 39
functions – C programming language 16

G

Gaelic language support 640
German language support 634
get-message options structure 81
GetMsgOpts parameter 350
GetWaitInterval field 40
glossary 691
Greek language support 650
GroupId field
  MQMD structure 146
  MQMDE structure 189
  MQPMR structure 234
GroupStatus field 82

H

handle scope 336, 382
handles 480
HardenGetBackout attribute 439
Hconn field 610
Hconn parameter
  MQBACK call 309
  MQBEGIN call 315
  MQCLOSE call 319
  MQCMIT call 327
  MQCONN call 336
  MQCONNX call 341
  MQDISC call 345
  MQGET call 349
  MQINQ call 363
  MQOPEN call 375
  MQPUT call 393
  MQPUT1 call 405
  MQSET call 415
  MQXCNVC call 614
  scope 336
header files
  C programming language 16
  Visual Basic programming language 26
Hebrew language support 652
Hobj parameter
  MQCLOSE call 319
  MQGET call 349
  MQINQ call 363
  MQOPEN call 382
  MQPUT call 393
  MQSET call 415
  scope 382
HTML (Hypertext Markup Language) 704
Hypertext Markup Language (HTML) 704

I

Icelandic language support 644
IGQPutAuthority attribute 477
IGQUserId attribute 478
InBuffer parameter 621
InBufferLength parameter 621
INCLUDE files – PL/I programming language 22
IndexType attribute 440
InhibitEvent attribute 479
InhibitGet attribute 441
InhibitPut attribute 441
InitiationQName attribute 442
InputItem field 40
IntAttrCount parameter
  MQINQ call 367
  MQSET call 416
IntAttrs parameter
  MQINQ call 367
  MQSET call 417
intra-group queuing 477, 478, 479
IntraGroupQueuing attribute 479
InvalidDestCount field
  MQOD structure 196
  MQPMO structure 212
Italian language support 638

J

Japanese language support 661, 662

K

Kanji language support 661, 662
Katakana language support 662
KnownDestCount field 197, 212
Korean language support 663

L

language compilers xix
Lao support 657
Latvian language support 648
LinkType field 40
Lithuanian language support 648
LocalEvent attribute 480
LTermOverride field 119

M

Macros 23
MatchOptions field 82
MaxHandles attribute 480
MaxMsgLength attribute
  queue 442
  queue manager 480
MaxPriority attribute 481
MaxQDepth attribute 443
MaxUncommittedMsgs attribute 481
message descriptor extension structure 185
message descriptor structure 125
message order 356, 399, 411
MFSMapName field 119
MQ_* values 547
MQ_CONNECT_TYPE environment variable 56
MQ_DATA_CONV_EXIT call 620
MQACT_* values 129
MQAT_* values
  ApplType
    attribute 464
    field 267
  PutApplType field 160
MQBACK call 309
MQBEGIN call 315
MQBND_* values 434
MQBO_* values 29
MQBO_DEFAULT 30
MQBO structure 29
MQBYTE 7
MQBYTEn 7
MQCA_* values 364, 416
MQCC_* values 489
MQCCSI_* values 131
MQCD_CLIENT_CONN_DEFAULT 54
MQCD_DEFAULT 54
MQCFUNC_* values 39
MQCGWI_* values 40
MQCHAR 8
MQCHARn 8
MQCI_* values 133
MQCIH_* values 43
MQCIH_DEFAULT 47
MQCIH structure 33
MQCLOSE call 319
MQCLT_* values 40
MQCMDL_* values 474
MQCMIT call 327
MQCNO_* values 54, 57
MQCNO_DEFAULT 58
MQCNO structure 51
MQCO_* values 320
MQCODL_* values 41
MQCONN call 333
MQCONNX call 341
MQCRC_* values 42
MQCT_* values 54
MQCUOWC_* values 45
MQDCC_* values 614
MQDH_* values 65
MQDH_DEFAULT 66
MQDH structure 61
MQDHF_* values 63
MQDISC call 345
MQDL_* values 438, 477
MQDLH_* values 75
MQDLH_DEFAULT 76
MQDLH structure 69
MQDXP_* values 612
MQDXP structure 607
MQEC_* values 110
MQEI_* values 136
MQENC_* values 133
MQEVR_* values
  AuthorityEvent attribute 471
  ChannelAutoDefEvent attribute 471
  InhibitEvent attribute 479
  LocalEvent attribute 480
  PerformanceEvent attribute 482
  QDepthHighEvent attribute 446
  QDepthLowEvent attribute 447
  QDepthMaxEvent attribute 448
  RemoteEvent attribute 484
  StartStopEvent attribute 485
MQFB_* values 74, 136
MQFMT_* values 140
MQGET call 349
MQGETANY call 359
MQGI_* values 147
MQGMO_* values 86, 111
MQGMO_DEFAULT 113
MQGMO structure 81
MQGS_* values 82

MQHC_* values 345
MQHCONN 8
MQHO_* values 319
MQHOBJ 8
MQIA_* values 364, 416
MQIAUT_* values 118
MQIAV_* values 368
MQICM_* values 118
MQIGQ_* values 479
MQIGQPA_* values 477
MQIIH_* values 120
MQIIH_DEFAULT 122
MQIIH structure 117
MQINQ call 363
MQISS_* values 120
MQIT_* values 440
MQITII_* values 121
MQITS_* values 121
MQLONG 9
MQMD_* values 176, 178
MQMD_DEFAULT 179
MQMD structure 125
MQMDE_* values 189
MQMDE_DEFAULT 190
MQMDE structure 185
MQMDEF_* values 188
MQMDS_* values 444
MQMF_* values 147
MQMI_* values 153
MQMO_* values 83
MQMT_* values 154
MQMTOK_* values 85
MQNC_* values 460
MQOD_* values 203
MQOD_DEFAULT 204
MQOD structure 193
MQOII_* values 256
MQOL_* values 156
MQOO_* values 376, 436
MQOPEN call 375
MQOR_DEFAULT 210
MQOR structure 209
MQOT_* values 200
MQPER_* values 156
MQPL_* values 482
MQPMO_* values 213, 227
MQPMO_DEFAULT 228
MQPMO structure 211
MQPMR structure 233
MQPMRF_* values 64, 222
MQPRI_* values 158
MQPTR 9
MQPUT call 393
MQPUT1 call 405
MQPUT1ANY call 412
MQPUTANY call 403
MQQA_* values
  InhibitGet attribute 441
  InhibitPut attribute 441
  Shareability attribute 453
MQQDT_* values 435
MQQSGD_* values 450, 461, 466
MQQSIE_* values 449
MQQT_* values 432, 451
MQRC_* values 139, 490
MQRFH_* values 238, 239, 240, 244, 248
MQRFH_DEFAULT 240
MQRFH structure 237
MQRFH2_DEFAULT 249
MQRFH2 structure 243
MQRL_* values 109
MQRMH_* values 257, 258
MQRMH_DEFAULT 258
MQRMH structure 251
MQRMHF_* values 255
MQRO_* values 166
MQRR_DEFAULT 264
MQRR structure 263
MQSCO_* values 453
MQSEG_* values 109
MQSeries publications 703
MQSET call 415
MQSID_* values 195
MQSIDT_* values 194
MQSP_* values 485
MQSS_* values 109
MQSYNC 423
MQTC_* values 454
MQTM_* values 269
MQTM_DEFAULT 270
MQTM structure 265
MQTMC_* values 274, 275
MQTMC2_DEFAULT 275
MQTMC2 structure 273
MQTT_* values 456
MQUS_* values 456
MQWI_* values 112
MQWIH_* values 281
MQWIH_DEFAULT 282
MQWIH structure 279
MQXC_* values 285
MQXCC_* values 287
MQXCNVC call 613
MQXDR_* values 609
MQXP_* values 288
MQXP structure 285
MQXQH_* values 295
MQXQH_DEFAULT 296
MQXQH structure 291
MQXR_* values 286
MQXT_* values 286
MQXUA_* values 287
MsgDeliverySequence attribute 444
MsgDesc field 294
MsgDesc parameter
  MQ_DATA_CONV_EXIT call 620
  MQGET call 349
  MQPUT call 393
  MQPUT1 call 405
MsgFlags field
  MQMD structure 147
  MQMDE structure 189
MsgId field
  MQMD structure 151
  MQPMR structure 235
MsgSeqNumber field
  MQMD structure 153
  MQMDE structure 189
MsgToken field 85, 281
MsgType field 154
multilingual language support 642

N

NameCount attribute 460
named constants – COBOL programming language 21
namelist attributes 459
NamelistDesc attribute 460
NamelistName attribute 460
Names attribute 461
NameValueCCSID field 245
NameValueData field 245
NameValueLength field 248
NameValueString field 238
NextTransactionId field 40
Norwegian language support 635
notational conventions
  C programming language 19
  COBOL programming language 22
  PL/I programming language 23
  S/370 assembler programming language 26
  Visual Basic programming language 27

O

ObjDesc parameter
  MQOPEN call 375
  MQPUT1 call 405
object descriptor structure 193
object record structure 209
ObjectInstanceId field 256
ObjectName field
  MQOD structure 197
  MQOR structure 209
ObjectQMgrName field
  MQOD structure 198
  MQOR structure 209
ObjectRecOffset field
  MQDH structure 64
  MQOD structure 199
ObjectRecPtr field 199
ObjectType field
  MQOD structure 200
  MQRMH structure 256
Offset field
  MQMD structure 155
  MQMDE structure 189
OpenInputCount attribute 444
OpenOutputCount attribute 445
Options field
  MQBO structure 29
  MQCNO structure 54
  MQGMO structure 86
  MQPMO structure 213
Options parameter
  MQCLOSE call 319
  MQOPEN call 376
  MQXCNVC call 614
ordering of messages 356, 399, 411
OriginalLength field
  MQMD structure 156
  MQMDE structure 189
OutBuffer parameter 621
OutBufferLength parameter 621
OutputDataLength field 40

P

parameters with undefined data types 17
PDF (Portable Document Format) 704
PerformanceEvent attribute 482
persistence 437
Persistence field 156
PL/I programming language
  INCLUDE files 22
  notational conventions 23
  structures 22
Platform attribute 482
PMQVOID 307
pointer data type – COBOL programming language 21
Portable Document Format (PDF) 704
Portuguese language support 643
PostScript format 705
Priority field 158
process definition attributes 463
ProcessDesc attribute 466
ProcessName
  attribute
    process definition 466
    queue 445
  field
    MQTM structure 268
    MQTMC2 structure 274
publications
  MQSeries 703
put-message options structure 211
put message record structure 233
PutApplName field
  MQDLH structure 72
  MQMD structure 159
PutApplType field
  MQDLH structure 73
  MQMD structure 160
PutDate field
  MQDLH structure 73
  MQMD structure 162
PutMsgOpts parameter
  MQPUT call 394
  MQPUT1 call 406
PutMsgRecFields field
  MQDH structure 64
  MQPMO structure 222
PutMsgRecOffset field
  MQDH structure 65
  MQPMO structure 223
PutMsgRecPtr field 224
PutTime field
  MQDLH structure 73
  MQMD structure 163

Q

QDepthHighEvent attribute 446
QDepthHighLimit attribute 446
QDepthLowEvent attribute 447
QDepthLowLimit attribute 447
QDepthMaxEvent attribute 447
QDesc attribute 448
QMgrDesc attribute 483
QMgrIdentifier attribute 483
QMgrName
  attribute 484
  field 274
QMgrName parameter
  MQCONN call 333
  MQCONNX call 341
QName
  attribute 448
  field
    MQTM structure 268
    MQTMC2 structure 274
QServiceInterval attribute 449
QServiceIntervalEvent attribute 449
QSGDisp attribute
  namelist 461, 466
  queue 450
QSGName attribute 484
QType attribute 451
queue, dynamic 375
queue attributes 427
queue-manager aliasing 427
queue manager attributes 469
queue-sharing group 198, 334, 484

R

reason codes
  alphabetic list 489
  numeric list 573
Reason field 41
  MQDLH structure 74
  MQDXP structure 610
  MQRR structure 263
Reason parameter
  MQBACK call 309
  MQBEGIN call 315
  MQCLOSE call 321
  MQCMIT call 327
  MQCONN call 336
  MQCONNX call 342
  MQDISC call 346
  MQGET call 351
  MQINQ call 369
  MQOPEN call 383
  MQPUT call 395
  MQPUT1 call 406
  MQSET call 417
  MQSYNC call 423
  MQXCNVC call 618
RecsPresent field
  MQDH structure 65
  MQOD structure 200
  MQPMO structure 224
reference message header structure 251
RemoteEvent attribute 484
RemoteQMgrName
  attribute 451
  field 294
RemoteQName
  attribute 452
  field 294
RemoteSysId field 41
RemoteTransId field 41
reply queue aliasing 427
ReplyToFormat field 41, 120
ReplyToQ field 164
ReplyToQMgr field 165
Report field
  MQMD structure 165
Report field *(continued)*
  using 593
report message conversion 605
RepositoryName attribute 484
RepositoryNamelist attribute 485
Reserved field
  MQIIH structure 120
  MQWIH structure 281
  MQXP structure 288
Reserved1 field 42, 108
Reserved2 field 42
Reserved3 field 42
Reserved4 field 42
ResolvedQMgrName field
  MQOD structure 201
  MQPMO structure 225
ResolvedQName field
  MQGMO structure 108
  MQOD structure 201
  MQPMO structure 225
response record structure 263
ResponseRecOffset field
  MQOD structure 202
  MQPMO structure 225
ResponseRecPtr field
  MQOD structure 202
  MQPMO structure 226
RetentionInterval attribute 452
return codes 489
ReturnCode field 42
ReturnedLength field 109
rules and formatting header structure 237
rules and formatting header structure version 2 243

S

scope, handles 336, 382
Scope attribute 453
SecurityScope field 120
Segmentation field 109
SegmentStatus field 109
SelectorCount parameter
  MQINQ call 363
  MQSET call 415
Selectors parameter
  MQINQ call 364
  MQSET call 415
ServiceName field 281
ServiceStep field 281
Shareability attribute 453
shared queue 198, 355
Signal1 field 110
Signal2 field 111
softcopy books 704
SourceBuffer parameter 617
SourceCCSID parameter 617
SourceLength parameter 617
Spanish language support 639
SrcEnvLength field 256
SrcEnvOffset field 256
SrcNameLength field 257
SrcNameOffset field 257
StartCode field 43
StartStopEvent attribute 485
StorageClass attribute 454
StrucId field
  MQBO structure 29

StrucId field *(continued)*
MQCIH structure 43
MQCNO structure 57
MQDH structure 65
MQDLH structure 75
MQDXP structure 612
MQGMO structure 111
MQIIH structure 120
MQMD structure 176
MQMDE structure 189
MQOD structure 203
MQPMO structure 227
MQRFH structure 239
MQRFH2 structure 248
MQRMH structure 257
MQTM structure 269
MQTMC2 structure 274
MQWIH structure 281
MQXP structure 288
MQXQH structure 295
StrucLength field
MQCIH structure 44
MQDH structure 65
MQIIH structure 120
MQMDE structure 189
MQRFH structure 240
MQRFH2 structure 248
MQRMH structure 257
MQWIH structure 281
structures – COBOL programming language 20
structures – PL/I programming language 22
supported language compilers xix
SupportPac 705
Swedish language support 637
syncpoint 485
SyncPoint attribute 485

T

TargetBuffer parameter 617
TargetCCSID parameter 617
TargetLength parameter 617
TaskEndStatus field 44
terminology xviii
terminology used in this book 691
Thai support 656
Timeout field 227
TranInstanceId field 121
TransactionId field 44
TransId parameter 423
transmission queue header structure 291
TranState field 121
trigger message structure 265
TriggerControl attribute 454
TriggerData
attribute 455
field
MQTM structure 269
MQTMC2 structure 274
TriggerDepth attribute 455
triggering 454
TriggerInterval attribute 486
TriggerMsgPriority attribute 455
TriggerType attribute 456
trusted application 55
Turkish language support 651
type styles in this book xxii

U

UCS-2 683
UK English language support 640
Ukrainian language support 649
Uncommitted messages 481
Unicode 683
UnknownDestCount field
MQOD structure 203
MQPMO structure 227
UOWControl field 45
Urdu support 655
US English language support 632
Usage attribute 456
use from C++ 19
UserData
attribute 467
field
MQTM structure 269
MQTMC2 structure 275
UserIdentifier field 176
UTF-8 683

V

Version field
MQBO structure 30
MQCIH structure 45
MQCNO structure 57
MQDH structure 66
MQDLH structure 75
MQDXP structure 612
MQGMO structure 111
MQIIH structure 121
MQMD structure 178
MQMDE structure 190
MQOD structure 203
MQPMO structure 227
MQRFH structure 240
MQRFH2 structure 248
MQRMH structure 258
MQTM structure 270
MQTMC2 structure 275
MQWIH structure 282
MQXP structure 288
MQXQH structure 295
Vietnamese language support 658
Visual Basic programming language
form files 26
header files 26
MQI calls 27
notational conventions 27
using data types 26

W

WaitInterval field 112
Windows Help 705
Windows products xviii

X

XmitQName attribute 457

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To make comments about the functions of IBM products or systems, talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom
- By fax:
  - From outside the U.K., after your international access code use 44–1962–870229
  - From within the U.K., use 01962–870229
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink™: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

© Copyright IBM Corp. 1994, 2000

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC33-1673-07

Spine information:

MQSeries®

MQSeries Application Programming Reference