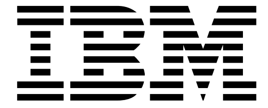
MQSeries for AS/400

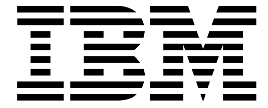


Application Programming Reference (RPG)

Version 4 Release 2

SC33-1957-00

MQSeries for AS/400



Application Programming Reference (RPG)

Version 4 Release 2

SC33-1957-00

Note!

Before using this information and the product it supports, be sure to read the general information under Appendix E, “Notices” on page 411.

First edition (February 1998)

This edition applies to the following product:

MQSeries for AS/400 Version 4 Release 2 and to any subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

At the back of this publication is a page titled “Sending your comments to IBM”. If you want to make comments, but the methods described are not available to you, please address them to:

IBM United Kingdom Laboratories,
Information Development,
Mail Point 095,
Hursley Park,
Winchester,
Hampshire,
England,
SO21 2JN

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1994, 1998. All rights reserved.**
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

About this book . . . vii
Who this book is for . . . vii
What you need to know to understand this book . . . vii
How to use this book . . . viii
Appearance of text in this book . . . viii
Terms used in this book . . . viii
Softcopy links . . . viii
MQSeries publications . . . ix
MQSeries cross-platform publications . . . ix
MQSeries platform-specific publications . . . xi
MQSeries Level 1 product publications . . . xii
Softcopy books . . . xii
MQSeries information available on the Internet . . . xiii
Related publications . . . xiv

New function in MQSeries for AS/400 V4R2 . . . xv
Additions to the Application Programming Reference (RPG), SC33-1957-00 . . . xv

Part 1. The programming interface . . . 1

Chapter 1. Data type descriptions – elementary . . . 3
Conventions used in the descriptions of data types . . . 3
Elementary data types . . . 3

Chapter 2. Data type descriptions – structures . . . 7
Conventions used in the descriptions of data types . . . 7
Language considerations . . . 8
MQDH – Distribution header . . . 13
MQDLH – Dead-letter (undelivered-message) header . . . 19
MQGMO – Get message options . . . 27
MQIIH – IMS header . . . 53
MQMD – Message descriptor . . . 59
MQMDE – Message descriptor extension . . . 104
MQOD – Object descriptor . . . 110
MQOR – Object records . . . 118
MQPMO – Put message options . . . 120
MQPMR – Put message records . . . 137
MQRMH –Reference message header . . . 140
MQRR – Response records . . . 149
MQTM – Trigger message . . . 151
MQTMC – Trigger message (character format) . . . 156
MQXQH –Transmission queue header . . . 159

Chapter 3. Call descriptions . . . 167
Conventions used in the call descriptions . . . 167
MQCLOSE – Close object . . . 169
MQCONN – Connect queue manager . . . 175
MQDISC – Disconnect queue manager . . . 180
MQGET – Get message . . . 183

© Copyright IBM Corp. 1994, 1998

MQINQ – Inquire about object attributes . . . 194
MQOPEN – Open object . . . 204
MQPUT – Put message . . . 217
MQPUT1 – Put one message . . . 227
MQSET – Set object attributes . . . 236

Chapter 4. Attributes of MQSeries objects . . . 243
Attributes for all queues . . . 243
Attributes for local queues . . . 246
Attributes for local definitions of remote queues . . . 260
Attributes for alias queues . . . 262
Attributes for process definitions . . . 262
Attributes for the queue-manager . . . 264

Chapter 5. Return codes . . . 275
Completion code . . . 275
Reason code . . . 275

Chapter 6. MQSeries constants . . . 321
List of constants . . . 321

Part 2. Building your application . . . 347

Chapter 7. Building your application . . . 349
MQSeries copy files . . . 349
Preparing your programs to run . . . 349
Syncpoints in MQSeries for AS/400 applications . . . 350
Syncpoints in CICS for AS/400 applications . . . 351

Part 3. MQSeries sample applications . . . 353

Chapter 8. Sample programs . . . 355
Features demonstrated in the sample programs . . . 356
Preparing and running the sample programs . . . 357
The Put sample program . . . 358
The Browse sample program . . . 359
The Get sample program . . . 360
The Request sample program . . . 360
The Echo sample program . . . 364
The Inquire sample program . . . 365
The Set sample program . . . 367
The Triggering sample programs . . . 368
Running the samples using remote queues . . . 369

Part 4. Appendixes . . . 371

Appendix A. Rules for validating MQI options . . . 373
MQOPEN . . . 373
MQPUT . . . 373
MQPUT1 . . . 374
MQGET . . . 374

MQCLOSE . . . 374

Appendix B. Machine encodings . . . 375
Binary-integer encoding . . . 375
Packed-decimal-integer encoding . . . 376
Floating-point encoding . . . 376
Constructing encodings . . . 377
Analyzing encodings . . . 377
Summary of machine architecture encodings . . . 378

Appendix C. Report options . . . 379
Structure of the report field . . . 379
Analyzing the report field . . . 381
Structure of the message-flags field . . . 382

Appendix D. Data-conversion . . . 385
Conversion processing . . . 385
Processing conventions . . . 386
Conversion of report messages . . . 390
MQDXP - Data conversion header . . . 391
MQDATACONVEXIT - Data conversion exit . . . 398
MQXCNVC - Convert characters . . . 403

Appendix E. Notices . . . 411
Programming interface information . . . 411
Trademarks . . . 412

Glossary of terms and abbreviations . . . 413

Index . . . 421

Tables

1. Elementary data types in ILE . . . 5
2. Elementary data types in OPM . . . 6
3. RPG COPY files . . . 9
4. Fields in MQDH . . . 13
5. Initial values of fields in MQDH . . . 17
6. Fields in MQDLH . . . 19
7. Initial values of fields in MQDLH . . . 25
8. Fields in MQGMO . . . 27
9. MQGET options relating to messages in groups and segments of logical messages . . . 40
10. Outcome when MQGET or MQCLOSE call not consistent with group and segment information . . . 42
11. Initial values of fields in MQGMO . . . 51
12. Fields in MQIIH . . . 53
13. Initial values of fields in MQIIH . . . 57
14. Fields in MQMD . . . 59
15. Initial values of fields in MQMD . . . 100

16. Fields in MQMDE . . . 104
17. Queue-manager action when MQMDE specified on MQPUT or MQPUT1 105
18. Initial values of fields in MQMDE . . . 108
19. Fields in MQOD . . . 110
20. Initial values of fields in MQOD . . . 116
21. Fields in MQOR . . . 118
22. Initial values of fields in MQOR . . . 118
23. Fields in MQPMO . . . 120
24. MQPUT options relating to messages in groups and segments of logical messages . . . 125
25. Outcome when MQPUT or MQCLOSE call not consistent with group and segment information . . . 127
26. Initial values of fields in MQPMO . . . 135
27. Fields in MQPMR . . . 137
28. Fields in MQRMH . . . 140
29. Initial values of fields in MQRMH . . . 146
30. Fields in MQRR . . . 149
31. Initial values of fields in MQRR . . . 149
32. Fields in MQTM . . . 151
33. Initial values of fields in MQTM . . . 154
34. Fields in MQTMC . . . 156
35. Initial values of fields in MQTMC . . . 157
36. Fields in MQXQH . . . 159
37. Initial values of fields in MQXQH . . . 163
38. Effect of MQCLOSE options on various types of object and queue . . . 171
39. Valid MQOPEN options for each queue type . . . 209
40. Attributes for all queues . . . 243
41. Attributes for local and model queues . . . 246
42. Attributes for local definitions of remote queues . . . 260
43. Attributes for process definitions . . . 262
44. Attributes for the queue manager . . . 265
45. Names of the sample programs . . . 355
46. Sample programs demonstrating use of the MQI . . . 357
47. Client/Server sample program details . . . 363
48. Summary of encodings for machine architectures . . . 378
49. Fields in MQDXP . . . 391

About this book

MQSeries for AS/400 Version 4 Release 2 is part of the IBM MQSeries set of products. It provides application programming services on the AS/400 platform 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 for AS/400 programming interface in the RPG programming language.
- Contains information on how to build an executable application.
- Contains descriptions of sample programs.

Note to users

There are two approaches that can be taken when using the MQI from within an RPG program:

1. Dynamic calls to the QMQM program interface. This method is available to OPM and ILE RPG programs.
2. Bound Calls to the MQI procedures. This method is available only to ILE RPG programs.

Using bound calls is generally the preferred method, particulary when the program is making repeated calls to the MQI, as it requires less resource. See “Coding the ILE RPG Bound Calls” on page 11 for further information.

If you want to use OPM you must use the call identifier, `QMQM, CID=MQxxx` where `xxx` is the name of the function call you are using, for example `GET`.

You should also note that certain elementary data types have different values in the two versions of the language. See Table 1 on page 5 and Table 2 on page 6 for the differences.

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 will 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 for AS/400, you need to know how to write programs in the RPG programming language.

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

© Copyright IBM Corp. 1994, 1998

How to use this book

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

The book is divided into three parts:

Part 1. The programming interface
Presents detailed reference information about the programming interface of MQSeries for AS/400 (known as the MQI). It describes the data types that the MQI calls use, the parameters and return codes for the calls, and the attributes of MQSeries for AS/400 objects. It also describes the values of constants you need to use when you write MQSeries for AS/400 programs, and the error messages that may be generated when you run your programs.

Part 2. Building your application
Describes how to build MQSeries for AS/400 programs.

Part 3. MQSeries for AS/400 sample applications
Describes the design of the sample applications that are provided with MQSeries for AS/400.

Appearance of text in this book

This book uses the following type styles:

MQOPEN	Example of the name of a call
CMPCOD	Example of the name of a parameter of a call
MQMD	Example of the name of a data type or structure
OOSETA	Example of the name of a value

Terms used in this book

All new terms that this book introduces are defined in the glossary. In the body of this book, the following shortened names are used for these products:

CICS The CICS for AS/400 product

Also, we use the following shortened name for this language compiler:

RPG Means the IBM ILE RPG for OS/400 compiler (5763-RG1)

Softcopy links

This book is linked to the other books in the MQSeries library. If you are using BookManager READ/MVS or BookManager READ/VM, you can view the other books directly from citations in this book by selecting the name of the book with your cursor and pressing the ENTER key.

You can also view descriptions of the following directly from this book by selecting the word or name with your cursor and pressing the ENTER key:

- Error messages
- Abend reason codes
- MQI calls
- MQSeries commands

MQSeries publications

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 V5.0
- MQSeries for AS/400 V4R2
- MQSeries for AT&T GIS UNIX V2.2
- MQSeries for Digital OpenVMS V2.2.1
- MQSeries for HP-UX V5.0
- MQSeries for MVS/ESA V1.2
- MQSeries for OS/2 Warp V5.0
- MQSeries for SINIX and DC/OSx V2.2
- MQSeries for SunOS V2.2
- MQSeries for Sun Solaris V5.0
- MQSeries for Tandem NonStop Kernel V2.2
- MQSeries Three Tier
- MQSeries for Windows V2.0
- MQSeries for Windows V2.1
- MQSeries for Windows NT V5.0

Any exceptions to this general rule are indicated. (Publications that support the MQSeries Level 1 products are listed in "MQSeries Level 1 product publications" on page xii. For a functional comparison of the Level 1 and Level 2 MQSeries products, see the *MQSeries Planning Guide*.)

MQSeries Brochure
The *MQSeries Brochure*, G511-1908, gives a brief introduction to the benefits of MQSeries. It is intended to support the purchasing decision, and describes some authentic customer use of MQSeries.

MQSeries: An Introduction to Messaging and Queuing
GC33-0805, describes briefly what MQSeries is, how it works, and how it can solve some classic interoperability problems. This book is intended for a more technical audience than the *MQSeries Brochure*.

MQSeries Planning Guide
The *MQSeries Planning Guide*, GC33-1349, describes some key MQSeries concepts, identifies items that need to be considered before MQSeries is installed, including storage requirements, backup and recovery, security, and migration from earlier releases, and specifies hardware and software requirements for every MQSeries platform.

MQSeries Intercommunication
The *MQSeries Intercommunication* book, SC33-1872, defines the concepts of distributed queuing and explains how to set up a distributed queuing network in a variety of MQSeries environments. In particular, it demonstrates how to (1) configure communications to and from a representative sample of MQSeries products, (2) create required MQSeries objects, and (3) create and configure MQSeries channels. The use of channel exits is also described.

MQSeries Clients
The *MQSeries Clients* book, GC33-1632, describes how to install, configure, use, and manage MQSeries client systems.

MQSeries System Administration
The *MQSeries System Administration* book, SC33-1873, supports day-to-day management of local and remote MQSeries objects. It includes topics such as security, recovery and restart, transactional support,problem determination, the dead-letter queue handler, and the MQSeries links for Lotus Notes**.It also includes the syntax of the MQSeries control commands.

This book applies to the following MQSeries products only:

- MQSeries for AIX V5.0
- MQSeries for HP-UX V5.0
- MQSeries for OS/2 Warp V5.0
- MQSeries for Sun Solaris V5.0
- MQSeries for Windows NT V5.0

MQSeries Command Reference
The *MQSeries Command Reference* , SC33-1369, contains the syntax of the MQSC commands, which are used by MQSeries system operators and administrators to manage MQSeries objects.

MQSeries Programmable System Management
The *MQSeries Programmable System Management* book, SC33-1482, provides both reference and guidance information for users of MQSeries events, programmable command formats (PCFs), and installable services.

MQSeries Messages
The *MQSeries Messages* book, GC33-1876, which describes "AMQ" messages issued by MQSeries, applies to these MQSeries products only:

- MQSeries for AIX V5.0
- MQSeries for HP-UX V5.0
- MQSeries for OS/2 Warp V5.0
- MQSeries for Sun Solaris V5.0
- MQSeries for Windows NT V5.0
- MQSeries for Windows V2.0
- MQSeries for Windows V2.1

This book is available in softcopy only.

MQSeries Application Programming Guide
The *MQSeries Application Programming Guide*, SC33-0807, provides guidance information for users of the message queue interface (MQI). It describes how to design, write, and build an MQSeries application. It also includes full descriptions of the sample programs supplied with MQSeries.

MQSeries Application Programming Reference
The *MQSeries Application Programming Reference*, SC33-1673, provides comprehensive reference information for users of the MQI. It includes: data-type descriptions; MQI call syntax; attributes of MQSeries objects; return codes; constants; and code-page conversion tables.

MQSeries Application Programming Reference Summary
The *MQSeries Application Programming Reference Summary*, SX33-6095, summarizes the information in the *MQSeries Application Programming Reference*manual.

MQSeries Using C++

MQSeries Using C++, SC33-1877, provides both guidance and reference information for users of the MQSeries C++ programming-language binding to the MQI. MQSeries C++ is supported by V5.0 of MQSeries for AIX, HP-UX, OS/2 Warp, Sun Solaris, and Windows NT, and by MQSeries clients supplied with those products and installed in the following environments:

- AIX
- HP-UX
- OS/2
- Sun Solaris
- Windows NT
- Windows 3.1
- Windows 95

MQSeries C++ is also supported by MQSeries for AS/400 V4R2.

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

MQSeries for AIX V5.0 Quick Beginnings, GC33-1867

MQSeries for AS/400

MQSeries for AS/400 Version 4 Release 2 Licensed Program Specifications, GC33-1958 (softcopy only)

MQSeries for AS/400 Version 4 Release 2 Administration Guide, GC33-1956

MQSeries for AS/400 Version 4 Release 2 Application Programming Reference (RPG), SC33-1957

MQSeries for AT&T GIS UNIX

SC33-1642

MQSeries for Digital OpenVMS

MQSeries for Digital OpenVMS System Management Guide, GC33-1791

MQSeries for HP-UX

MQSeries for HP-UX V5.0 Quick Beginnings, GC33-1869

MQSeries for MVS/ESA

MQSeries for MVS/ESA Version 1 Release 2 Licensed Program Specifications, GC33-1350

MQSeries for MVS/ESA Version 1 Release 2 Program Directory

MQSeries for MVS/ESA Version 1 Release 2 System Management Guide, SC33-0806

MQSeries for MVS/ESA Version 1 Release 2 Messages, GC33-0819

MQSeries for MVS/ESA Version 1 Release 2 Problem Determination Guide, GC33-0808

MQSeries for OS/2 Warp

MQSeries for OS/2 Warp V5.0 Quick Beginnings, GC33-1868

MQSeries link for R/3

MQSeries link for R/3 Version 1.0 User's Guide, GC33-1934

MQSeries for SINIX and DC/OSx

MQSeries for SINIX and DC/OSx System Management Guide, GC33-1768

MQSeries for SunOS

MQSeries for SunOS System Management Guide, GC33-1772

MQSeries for Sun Solaris

MQSeries for Sun Solaris V5.0 Quick Beginnings, GC33-1870

MQSeries for Tandem NonStop Kernel

MQSeries for Tandem NonStop Kernel Version 2.2 System Management Guide, GC33-1893

MQSeries Three Tier

MQSeries Three Tier Administration Guide, SC33
MQSeries Three Tier Reference Summary, SX33-60
MQSeries Three Tier Application Design, SC33-1
MQSeries Three Tier Application Programming, S

MQSeries for Windows

MQSeries for Windows User's Guide, GC33-1822

MQSeries for Windows Version 2.1 User's Guide, GC33-1965

MQSeries for Windows NT

MQSeries for Windows NT V5.0 Quick Beginnings, GC33-1871

MQSeries Level 1 product publications

For information about the MQSeries Level 1 products, see the following publications:

MQSeries: Concepts and Architecture, GC3

MQSeries Version 1 Products for UNIX Operating Systems Messages and Codes, SC33-1754

MQSeries for Digital VMS VAX Version 1.5 User's Guide, SC33-1144

MQSeries for SCO UNIX Version 1.4 User's Guide, SC33-1378

MQSeries for UnixWare Version 1.4.1 User's Guide, SC33-1379

MQSeries for VSE/ESA Version 1 Release 4 Licensed Program Specifications, GC33-1483

MQSeries for VSE/ESA Version 1 Release 4 User's Guide, SC33-1142

Softcopy books

Most of the MQSeries books are supplied in both hardcopy and softcopy formats.

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 products, including all MQSeries V5.0 products. Books in PostScript format can be printed on a PostScript printer or viewed with a suitable viewer.

HTML format

The MQSeries documentation is provided in HTML format with these MQSeries products:

- MQSeries for AIX V5.0
- MQSeries for HP-UX V5.0
- MQSeries for OS/2 Warp V5.0
- MQSeries for Sun Solaris V5.0
- MQSeries for Windows NT V5.0

The MQSeries books are also available from the MQSeries software-server home page at:

```
http://www.software.ibm.com/ts/mqseries/
```

Information Presentation Facility (IPF) format

In the OS/2 environment, the MQSeries documentation is supplied in IBM IPF format on the MQSeries product CD-ROM.

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

MQSeries URL

The URL of the MQSeries product family home page is:

```
http://www.software.ibm.com/mqseries/
```

Related publications

ILE RPG for AS/400 Programmer's Guide, SC09-2507
ILE RPG for AS/400 Reference, SC09-2508

New function in MQSeries for AS/400 V4R2

MQSeries for AS/400 Version 4 Release 2 provides the following new function:

- Distribution lists, which allow a single message to be put to multiple queues using a single MQPUT or MQPUT1 call.
- Reference messages, which allow the transfer of large amounts of message data between nodes.
- Message segmentation and grouping of messages.
- Fast nonpersistent messages.
- Channel heartbeats. Heartbeat flows are passed from a sending MCA when there are no messages on the transmission queue, allowing the receiving MCA the opportunity to quiesce the channel.
- Automatic creation of channel definitions for receiver and server-connection channels.
- Static bindings for the ILE RPG programming language
- Support for MQI applications written in C++ .
- Chained exits.
- Improved triggering rules.

Additions to the Application Programming Reference (RPG), SC33-1957-00

Changes to the book include:

- Addition of distribution lists, which include the:
  - MQDH data type structure
  - MQOR data type structure
  - MQPMR data type structure
  - MQRR data type structure
- Addition of message groups and segmentation of large messages which includes the MQMDE data type structure.
- Addition of reference messages which includes the MQRMH data type structure.
- Addition of static bindings for ILE.

© Copyright IBM Corp. 1994, 1998

Part 1. The programming interface

© Copyright IBM Corp. 1994, 1998

Chapter 1. Data type descriptions – elementary

This chapter describes the elementary data types used by the MQI.

The elementary data types are:

- 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

Conventions used in the descriptions of data types

For each elementary data type, this chapter gives a description of its usage, in a form that is independent of the programming language. This is followed by typical declarations in both ILE and OPM versions of the RPG programming language. The definitions of elementary data types are included here to provide consistency. RPG uses ‘I’ or ‘D’ specifications where working fields can be declared using whatever attributes you need. You can, however, do this in the calculation specifications where the field is used.

To use the elementary data types, you create:

- A /COPY member containing all the data types, or
- An external data structure (PF) containing all the data types. You then need to specify your working fields with attributes ‘LIKE’ the appropriate data type field.

The benefits of the second option are that the definitions can be used as a ‘FIELD REFERENCE FILE’ for other AS/400 objects. If an MQ data type definition changes, it is a relatively simple matter to recreate these objects.

Elementary data types

All of the other data types described in this chapter equate either directly to these elementary data types, or to aggregates of these elementary data types (arrays or structures).

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.

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.

© Copyright IBM Corp. 1994, 1998

MQBYTEn – String of n bytes

Each MQBYTEn data type represents a string of *n* bytes, where *n* can take one of the following values:

16, 24, 32, or 64

Each byte is described by the MQBYTE 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 nulls to fill the string.

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

Constants are available that define the lengths of byte string fields; see “LN* (Lengths of character string and byte fields)” on page 321.

MQCHAR – character

The MQCHAR data type represents a single character. The coded character set identifier of the character is that of the queue manager (see the *CodedCharSetId* attribute on page 252). No special alignment is required.

Note: Application message data specified on the MQGET, MQPUT, and MQPUT1 calls is described by the MQBYTE data type, not the MQCHAR data type.

MQCHARn – String of n characters

Each MQCHARn data type represents a string of *n* characters, where *n* can take one of the following values:

4, 8, 12, 16, 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.

Constants are available that define the lengths of character string fields; see “LN* (Lengths of character string and byte fields)” on page 321.

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 its natural 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 its natural 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, aligned on its natural boundary.

Elementary data types - ILE RPG

Table 1. Elementary data types in ILE

Data type	Representation
MQBYTE	A 1-byte alphanumeric field.
MQBYTE16	A 16-byte alphanumeric field.
MQBYTE24	A 24-byte alphanumeric field.
MQBYTE32	A 32-byte alphanumeric field.
MQBYTE64	A 64-byte alphanumeric field.
MQCHAR	A 1-byte alphanumeric field.
MQCHAR4	A 4-byte alphanumeric field.
MQCHAR8	An 8-byte alphanumeric field.
MQCHAR12	A 12-byte alphanumeric field.
MQCHAR16	A 16-byte alphanumeric field.
MQCHAR20	A 20-byte alphanumeric field.
MQCHAR28	A 28-byte alphanumeric field.
MQCHAR32	A 32-byte alphanumeric field.
MQCHAR48	A 48-byte alphanumeric field.
MQCHAR64	A 64-byte alphanumeric field.
MQCHAR128	A 128-byte alphanumeric field.
MQCHAR256	A 256-byte alphanumeric field.
MQHCONN	A 10-digit signed integer.
MQHOBJ	A 10-digit signed integer.
MQLONG	A 10-digit signed integer.
PMQLONG	A 10-digit signed integer.

Elementary data types – OPM

Table 2. Elementary data types in OPM

Data type	Representation
MQBYTE	A 1-byte alphanumeric field.
MQBYTE16	A 16-byte alphanumeric field.
MQBYTE24	A 24-byte alphanumeric field.
MQBYTE32	A 32-byte alphanumeric field.
MQBYTE64	A 64-byte alphanumeric field.
MQCHAR	A 1-byte alphanumeric field.
MQCHAR4	A 4-byte alphanumeric field.
MQCHAR8	An 8-byte alphanumeric field.
MQCHAR12	A 12-byte alphanumeric field.
MQCHAR16	A 16-byte alphanumeric field.
MQCHAR20	A 20-byte alphanumeric field.
MQCHAR28	A 28-byte alphanumeric field.
MQCHAR32	A 32-byte alphanumeric field.
MQCHAR48	A 48-byte alphanumeric field.
MQCHAR64	A 64-byte alphanumeric field.
MQCHAR128	A 128-byte alphanumeric field.
MQCHAR256	A 256-byte alphanumeric field.
MQHCONN	**When a structure subfield:** A 4-byte signed binary integer. **When a call parameter:** A 9-digit signed packed-decimal integer.
MQHOBJ	**When a structure subfield:** A 4-byte signed binary integer. **When a call parameter:** A 9-digit signed packed-decimal integer.
MQLONG	**When a structure subfield:** A 4-byte signed binary integer. **When a call parameter:** A 9-digit signed packed-decimal integer.
PMQLONG	**When a structure subfield:** A 4-byte signed binary integer. **When a call parameter:** A 9-digit signed packed-decimal integer.

Chapter 2. Data type descriptions – structures

This chapter describes the structure data types used by the MQI, which are:

MQGMO	– Get-message options
MQMD	– Message descriptor
MQMDE	– Message descriptor extension
MQOD	– Object descriptor
MQOR	– Object record
MQPMO	– Put-message options
MQPMR	– Put message record
MQRR	– Response record

The MQI also uses the following structure data types, which are included in this chapter for completeness, but they are not part of the application programming interface.

MQDH	– Distribution header
MQDLH	– Dead-letter (undelivered-message) header
MQIIH	– IMS bridge header (MVS/ESA only)
MQTM	– Trigger message
MQTMC	– Trigger message (character format)
MQXQH	– Transmission queue header

Note: The MQDXP – data conversion exit parameter structure is in Appendix D, "Data-conversion" on page 385, together with the associated data conversion calls.

Conventions used in the descriptions of data types

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

Structure name
The name of the structure, followed by a brief description of the purpose of the structure.

Fields
For each field, the name is followed by its elementary data type in parentheses ( ); for example:

Version (10-digit signed integer)

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, GMSIDV. A set of constants having the same prefix is shown using the * character, for example: IA*.

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.

© Copyright IBM Corp. 1994, 1998

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.

ILE declaration
Typical declaration of the structure in ILE.

OPM declaration
Typical declaration of the structure in OPM.

Language considerations

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

COPY files

Various COPY files are provided as part of the definition of the message queue interface (MQI), to assist with the writing of RPG application programs that use message queuing. There are two sets of COPY files for ILE RPG, and one set for OPM RPG:

- COPY files with names ending with the letter “G” are for use with programs that use static linkage. These COPY files can be used only by ILE RPG programs.

  The COPY files reside in QRPGLESRC in the QMQM library.

- COPY files with names ending with the letter “R” are for use with programs that use dynamic linkage. Two versions of these COPY files are provided:
  - One version is for use with ILE RPG programs. These COPY files reside in QRPGLESRC in the QMQM library.
  - The other version is for use with OPM RPG programs. These COPY files reside in QRPGSRC in the QMQM library.

For each set of COPY files, there are two files containing named constants, and one file for each of the structures. The COPY files are summarized in Table 3 on page 9.

Table 3. RPG COPY files

Filename (static linkage)	Filename (dynamic linkage)	Contents
CMQDHG	CMQDHR	Distribution header structure
CMQDLHG	CMQDLHR	Dead-letter (undelivered-message) header structure
CMQDXPG	CMQDXPR	Data-conversion-exit parameter structure
CMQGMOG	CMQGMOR	Get-message options structure
CMQIIHG	CMQIIHR	IMS information header structure
CMQMDG	CMQMDR	Message descriptor structure
CMQMDEG	CMQMDER	Message descriptor extension structure
CMQODG	CMQODR	Object descriptor structure
CMQORG	CMQORR	Object record structure
CMQPMOG	CMQPMOR	Put-message options structure
CMQRRG	CMQRRR	Response record structure
CMQTMG	CMQTMR	Trigger-message structure
CMQTMCG	CMQTMCR	Trigger message structure (character format)
CMQXQHG	CMQXQHR	Transmission-queue header structure
CMQG	CMQR	Named constants for main MQI
CMQXG	CMQXR	Named constants for data-conversion exit

Calls

In Chapter 3, “Call descriptions” on page 167, the calls are described using their individual names. In RPG using dynamic linkage, all calls are made to the single name **QMQM**, and the particular function required is specified by coding an additional parameter which precedes the normal parameters for that call. The following named constants may be used for this additional parameter, in order to identify the function required:

Named constant	Function required
MQCLOS	Close object.
MQCONN	Connect queue manager.
MQDISC	Disconnect queue manager.
MQGET	Get message.
MQINQ	Inquire about object attributes.
MQOPEN	Open object.
MQPUT	Put message.
MQPUT1	Put one message.
MQSET	Set object attributes.

These constants have names which are the same as the calls they identify, with the exception of the constant for the MQCLOSE call, which is abbreviated to MQCLOS.

Structures

With the exception of the MQTMC structure, all MQ structures are defined with initial values for the fields. These initial values are defined in the relevant table for each structure.

The structure declarations do not contain **DS** statements. This allows the application to declare either a single data structure or a multiple-occurrence data structure, by coding the **DS** statement and then using the **/COPY** statement to copy in the remainder of the declaration:

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7
I* Declare an MQMD data structure with 5 occurrences
IMYMD        DS                          5
I/COPY CMQMDR
```

Notational conventions

The sections that follow show how the:

- Calls should be invoked
- Parameters should be declared
- Various data types should be declared.

In a number of cases, parameters are arrays or character strings whose size is not fixed. For these, a lower case “n” is used to represent a numeric constant. When the declaration for that parameter is coded, the “n” must be replaced by the numeric value required.

MQI procedures

When using the ILE bound calls, you must reference the MQI procedures when you create your program. All the required procedures exist in the service program QMQM/AMQZSTUB, so the command to create your program should take the following basic form:

```
CRTPGM  PGM(MYPROGRAM) BNDSRVPGM(QMQM/AMQZSTUB)
```

MQI call parameters.

Many parameters passed to the MQI can have more than one concurrent function. This is because the integer value passed is often tested on the setting of individual bits within the field, and not on its total value. This allows you to ‘add’ several functions together and pass them as a single parameter.

Named Constants

There are a large number of different integer and character values that provide data interchange between your application program and the MQI. To facilitate a more readable and consistent approach to using these values, they have all been allocated named constants.

You should refer to these constants at all times, and not the values they represent, because this ensures readibility and easier migration of code from one platform to another.

If the value of any of these constants should change, you will only need to recompile your program to incorporate the changes.

All Named Constants are available by referencing the COPY members.

Commitment control

The MQI syncpoint functions, MQCMIT and MQBACK are not supported on the AS/400. MQSeries for AS/400 supports the native commitment control operation codes for each language.

For OPM RPG you can use the operation codes COMIT and ROLBK, and for ILE RPG you can use the operation codes COMMIT and ROLBK.

Coding the ILE RPG Bound Calls

MQI ILE Procedures are contained in Service program QMQM/AMQZSTUB and are named as follows:

MQCONN
MQDISC
MQOPEN
MQCLOSE
MQGET
MQPUT
MQPUT1
MQINQ
MQSET

To use these procedures you need to:

1. Define the external procedures in your 'D' specifications. These are all available within the Named Constant COPY file member CMQG.
2. Use the CALLP operation code to call the procedure along with its parameters.

For example the MQOPEN call requires the inclusion of the following code:

```
D*****************************************************************
D**  MQOPEN Call -- Open Object  (From COPY file CMQG)          **
D*****************************************************************
D*
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQOPEN           PR                  EXTPROC('MQOPEN')
D* Connection handle
D HCONN                         10I 0 VALUE
D* Object descriptor
D OBJDSC                       224A
D* Options that control the action of MQOPEN
D OPTS                          10I 0 VALUE
D* Object handle
D HOBJ                          10I 0
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CMPCOD
D REASON                        10I 0
D*
```

To call the procedure, after initializing the various parameters, you need the following code:

```
...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+....8
    C                   CALLP     MQOPEN(HCONN : MQOD : OPTS : HOBJ :
    C                             CMPCOD : REASON)
```

Here, the structure MQOD is defined using the COPY member CMQODG which breaks it down into its components.

Coding Dynamic calls to the MQI (OPM and ILE)

To use the MQI through dynamic calls to QMQM, you require the following code. The example is again MQOPEN:

```
...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+....8
    C                   Z-ADD     MQOPEN        CID
    C                   CALL      'QMQM'
    C                   PARM                    CID                9 0
    C                   PARM                    HCONN              9 0
    C                   PARM                    MQOD
    C                   PARM                    OPTS               9 0
    C                   PARM                    HOBJ               9 0
    C                   PARM                    CMPCOD             9 0
    C                   PARM                    REASON             9 0
     *
```

Here, the structure MQOD is defined using the COPY member CMQODR which splits it into its components.

MQDH – Distribution header

The following table guides you to the appropriate page for each field.

Table 4. Fields in MQDH

This field...	Describes...	See page ...
DHSID	Structure identifier	14
DHVER	Structure version number	14
DHLEN	Total length of structure	14
DHENC	Data encoding	15
DHCSI	Coded character-set identifier	15
DHFMT	Format name	15
DHFLG	General flags	15
DHPRF	Flags for put message record fields	15
DHCNT	Number of destinations	16
DHORO	Offset of first MQOR record	16
DHPRO	Offset of first MQPMR record	16

The MQDH structure describes the data that is present in a message on a transmission queue when that message is a distribution-list message (that is, the message is being sent to multiple destination queues). 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.

This structure is supported in the following environments: AIX, DOS client, HP-UX, OS/2, OS/400, Sun Solaris, Windows client, Windows NT.

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
- 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 RC2135.

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 on page 250). 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

DHSID (4-byte character string)
: Structure identifier.

  The value must be:

  DHSID V
  : Identifier for distribution header structure.

  The initial value of this field is DHSIDV.

DHVER (10-digit signed integer)
: Structure version number.

  The value must be:

  DHVER1
  : Version number for distribution header structure.

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

  DHVERC
  : Current version of distribution header structure.

  The initial value of this field is DHVER1.

DHLEN (10-digit signed integer)
: 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 *DHLEN*, 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.

DHENC (10-digit signed integer)
Encoding of message data.

The initial value of this field is 0.

DHCSI (10-digit signed integer)
Coded character-set identifier of message data.

The initial value of this field is 0.

DHFMT (8-byte character string)
Format name of message data.

The initial value of this field is FMNONE.

DHFLG (10-digit signed integer)
General flags.

The following flag can be specified:

DHFNEW
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 *PRMID* 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 DHFNEW 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 *PRMID* field.
- The *MDMID* field in MQMD is MINONE, or the *PMOPT* field in MQPMO includes PMNMID

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

DHFNON
No flags.

This constant indicates that no flags have been specified. DHFNON 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 DHFNON.

DHPRF (10-digit signed integer)
Flags indicating which MQPMR fields are present.

Zero or more of the following flags can be specified:

PFMID
Message-identifier field is present.

PFCID
: Correlation-identifier field is present.

PFGID
: Group-identifier field is present.

PFFB
: Feedback field is present.

PFACC
: Accounting-token field is present.

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

PFNONE
: No put-message record fields are present.

  PFNONE 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 PFNONE.

DHCNT (10-digit signed integer)
: Number of object records present.

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

  The initial value of this field is 0.

DHORO (10-digit signed integer)
: 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 *DHCNT* 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 *DHLEN* field.

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

  The initial value of this field is 0.

DHPRO (10-digit signed integer)
: 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 *DHCNT* 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 *DHLEN* field.

  Put message records are optional; if no records are provided, *DHPRO* is zero, and *DHPRF* has the value PFNONE.

  The initial value of this field is 0.

Table 5. Initial values of fields in MQDH

Field name	Name of constant	Value of constant
DHSID	DHSIDV	'DHbb' (See note 1)
DHVER	DHVER1	1
DHLEN	None	0
DHENC	None	0
DHCSI	None	0
DHFMT	FMNONE	'bbbbbbbb'
DHFLG	DHFNON	0
DHPRF	None	0
DHCNT	None	0
DHORO	None	0
DHPRO	None	0

Notes:

1. The symbol ‘b’ represents a single blank character.

RPG declaration (ILE)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQDH Structure
D*
D* Structure identifier
D  DHSID                  1      4
D* Structure version number
D  DHVER                  5      8I 0
D* Length of MQDH structure plus following records
D  DHLEN                  9     12I 0
D* Encoding of message data
D  DHENC                 13     16I 0
D* Coded character-set identifier of message data
D  DHCSI                 17     20I 0
D* Format name of message data
D  DHFMT                 21     28
D* General flags
D  DHFLG                 29     32I 0
D* Flags indicating which MQPMR fields are present
D  DHPRF                 33     36I 0
D* Number of object records present
D  DHCNT                 37     40I 0
D* Offset of first object record from start of MQDH
D  DHORO                 41     44I 0
D* Offset of first put message record from start of MQDH
D  DHPRO                 45     48I 0
```

RPG declaration (OPM)

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7..
I* MQDH Structure
I*
I* Structure identifier
I                                       1   4 DHSID
I* Structure version number
I                                   B   5   80DHVER
I* Length of MQDH structure plus following records
I                                   B   9  120DHLEN
I* Encoding of message data
I                                   B  13  160DHENC
I* Coded character-set identifier of message data
I                                   B  17  200DHCSI
I* Format name of message data
I                                      21  28 DHFMT
I* General flags
I                                   B  29  320DHFLG
I* Flags indicating which MQPMR fields are present
I                                   B  33  360DHPRF
I* Number of object records present
I                                   B  37  400DHCNT
I* Offset of first object record from start of MQDH
I                                   B  41  440DHORO
I* Offset of first put message record from start of MQDH
I                                   B  45  480DHPRO
```

MQDLH – Dead-letter (undelivered-message) header

The following table guides you to the appropriate page for each field.

Table 6. Fields in MQDLH

This field...	Describes...	See page ...
DLSID	Structure identifier	21
DLVER	Structure version number	21
DLREA	Reason message arrived on dead-letter queue	21
DLDQ	Name of original destination queue	22
DLDM	Name of original destination queue manager	22
DLENC	Original data encoding	22
DLCSI	Original coded character set identifier	23
DLFMT	Original format name	23
DLPAT	Type of application that put message on dead-letter queue	23
DLPAN	Name of application that put message on dead-letter queue	23
DLPD	Date when message was put on dead-letter queue	24
DLPT	Time when message was put on dead-letter queue	24

The MQDLH structure describes the information that is prefixed to 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.

Special processing is done 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.

This structure is *not* supported in the following environments: 16-bit Windows, 32-bit Windows.

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 check that an MQDLH structure is present, or that valid values have been specified for the fields.

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.

When a message is put on the dead-letter queue, all of the fields in the message descriptor MQMD should be copied from those in the original message descriptor (if there is one), with the exception of the following:

- The *MDCSI* and *MDENC* fields should be set to whatever character set and encoding are used for fields in the MQDLH structure.
- The *MDFMT* field should be set to FMDLH to indicate that the data begins with a MQDLH structure.
- The context fields:

  MDUID
  MDACC
  MDAID
  MDPAT
  MDPAN
  MDPD
  MDPT
  MDAOD

  should be set by using a context option appropriate to the nature of the program:

  – A program putting on the dead-letter queue a message that is not related to any preceding message should use the PMDEFC option; this causes the queue manager to set all of the context fields in the message descriptor to their default values.

  – A program putting on the dead-letter queue a message it has just received should use the PMPASA option, in order to preserve the original context information.

  – A program putting on the dead-letter queue a *reply* to a message it has just received should use the PMPASI option; this preserves the identity information but sets the origin information to be that of the server.

  – A message channel agent putting on the dead-letter queue a message it received from its communication channel should use the PMSETA option, to preserve the original context information.

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

- The *DLCSI*, *DLENC* and *DLFMT* fields should be set to the values that describe the application message data that follows the MQDLH structure—usually the values from the original message descriptor.
- The context fields *DLPAT*, *DLPAN*, *DLPD*, and *DLPT* 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.

Character data in the MQDLH structure should be in the character set defined by the *MDCSI* field of the message descriptor. Numeric data in the MQDLH structure should be in the data encoding defined by the *MDENC* field of the message descriptor. 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.

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 *MDFMT* field in the message descriptor MQMD; if the field has the value FMDLH, 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

DLSID (4-byte character string)
Structure identifier.

The value must be:

DLSIDV
Identifier for dead-letter header structure.

The initial value of this field is DLSIDV.

DLVER (10-digit signed integer)
Structure version number.

The value must be:

DLVER1
Version number for dead-letter header structure.

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

DLVERC
Current version of dead-letter header structure.

The initial value of this field is DLVER1.

DLREA (10-digit signed integer)
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 FB*or RC*values (for example, RC2053). See the description of the *MDFB* field on page 74 for details of the common FB* values that can occur.

If the value is in the range FBIFST through FBILST, the actual IMS error code can be determined by subtracting FBIERR from the value of the *DLREA* field.

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

FBABEG
: Application cannot be started.

  An application processing a trigger message was unable to start the application named in the *TMAI* field of the trigger message (see “MQTM – Trigger message” on page 151).

FBTM
: MQTM structure not valid or missing.

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

FBATYP
: Application type error.

  An application processing a trigger message was unable to start the application because the *TMAT* field of the trigger message is not valid (see “MQTM – Trigger message” on page 151).

FBXQME
: 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 RCNONE.

DLDQ (48-byte character string)
: 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 LNQN. The initial value of this field is 48 blank characters.

DLDM (48-byte character string)
: 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 LNQMN. The initial value of this field is 48 blank characters.

DLENC (10-digit signed integer)
: Original data encoding.

  This specifies the data encoding used for numeric data in the original message. It applies to the message data which *follows* the MQDLH structure; it does not apply to numeric data in the MQDLH structure itself.

  When an MQDLH structure is prefixed to the message data, the original data encoding should be preserved by copying it from the *MDENC* field in the message descriptor MQMD to the *DLENC* field in the MQDLH structure.

The *MDENC* field in the message descriptor should then be set to the value appropriate to the numeric data in the MQDLH structure.

The value ENNAT can be used for the *DLENC* field in both the MQDLH and MQMD structures.

The initial value of this field is 0.

DLCSI (10-digit signed integer)
Original coded character set identifier.

This specifies the coded character set identifier of character data in the original message. It applies to the message data which *follows* the MQDLH structure; it does not apply to character data in the MQDLH structure itself.

When an MQDLH structure is prefixed to the message data, the original coded character set identifier should be preserved by copying it from the *MDCSI* field in the message descriptor MQMD to the *DLCSI* field in the MQDLH structure. The *MDCSI* field in the message descriptor should then be set to the value appropriate to the character data in the MQDLH structure.

The value CSQM can be used for the *MDCSI* field in the MQMD structure, but should not be used for the *DLCSI* field in the MQDLH structure, as the queue manager does not replace the value CSQM in the latter field by the value that applies to the queue manager.

The initial value of this field is 0.

DLFMT (8-byte character string)
Original format name.

This is the format name of the application data in the original message. It applies to the message data which *follows* the MQDLH structure; it does not apply to the MQDLH structure itself.

When an MQDLH structure is prefixed to the message data, the original format name should be preserved by copying it from the *MDFMT* field in the message descriptor MQMD to the *DLFMT* field in the MQDLH structure. The *MDFMT* field in the message descriptor should then be set to the value FMDLH.

The length of this field is given by LNFMT. The initial value of this field is FMNONE.

DLPAT (10-digit signed integer)
Type of application that put message on dead-letter (undelivered-message) queue.

This field has the same meaning as the *MDPAT* field in the message descriptor MQMD (see page 89 for details).

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

The initial value of this field is 0.

DLPAN (28-byte character string)
Name of application that put message on dead-letter (undelivered-message) queue.

The format of the name depends on the *DLPAT* field. See, also, the description of the *MDPAN* field in MQMD on page 89.

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

The length of this field is given by LNPAN. The initial value of this field is 28 blank characters.

DLPD (8-byte character string)
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 *DLPD* and *DLPT* fields, subject to the system clock being set accurately to GMT.

The length of this field is given by LNPDAT. The initial value of this field is 8 blank characters.

DLPT (8-byte character string)
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 *DLPT*. This happens when leap seconds are inserted into the global time standard.

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

The length of this field is given by LNPTIM. The initial value of this field is 8 blank characters.

Table 7. Initial values of fields in MQDLH

Field name	Name of constant	Value of constant
DLSID	DLSIDV	'DLHb' (See note 1)
DLVER	DLVER1	1
DLREA	RCNONE	0
DLDQ	None	Blanks
DLDM	None	Blanks
DLENC	None	0
DLCSI	None	0
DLFMT	FMNONE	'bbbbbbbb'
DLPAT	None	0
DLPAN	None	Blanks
DLPD	None	Blanks
DLPT	None	Blanks

Notes:

1. The symbol 'b' represents a single blank character.

RPG declaration (ILE)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQDLH Structure
D*
D* Structure identifier
D  DLSID                  1      4
D* Structure version number
D  DLVER                  5      8I 0
D* Reason message arrived on dead-letter (undelivered-message)
D* queue
D  DLREA                  9     12I 0
D* Name of original destination queue
D  DLDQ                  13     60
D* Name of original destination queue manager
D  DLDM                  61    108
D* Original data encoding
D  DLENC                109    112I 0
D* Original coded character set identifier
D  DLCSI                113    116I 0
D* Original format name
D  DLFMT                117    124
D* Type of application that put message on dead-letter
D* (undelivered-message) queue
D  DLPAT                125    128I 0
D* Name of application that put message on dead-letter
D* (undelivered-message) queue
D  DLPAN                129    156
D* Date when message was put on dead-letter (undelivered-message)
D* queue
D  DLPD                 157    164
D* Time when message was put on the dead-letter
D* (undelivered-message) queue
```

```
D  DLPT                 165    172
```

RPG declaration (OPM)

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7..
I* MQDLH Structure
I*
I* Structure identifier
I                                         1   4 DLSID
I* Structure version number
I                                     B   5   80DLVER
I* Reason message arrived on dead-letter (undelivered-message)
I* queue
I                                     B   9  120DLREA
I* Name of original destination queue
I                                        13  60 DLDQ
I* Name of original destination queue manager
I                                        61 108 DLDM
I* Original data encoding
I                                     B 109 1120DLENC
I* Original coded character set identifier
I                                     B 113 1160DLCSI
I* Original format name
I                                       117 124 DLFMT
I* Type of application that put message on dead-letter
I* (undelivered-message) queue
I                                     B 125 1280DLPAT
I* Name of application that put message on dead-letter
I* (undelivered-message) queue
I                                       129 156 DLPAN
I* Date when message was put on dead-letter (undelivered-message)
I* queue
I                                       157 164 DLPD
I* Time when message was put on the dead-letter
I* (undelivered-message) queue
I                                       165 172 DLPT
```

MQGMO – Get message options

The following table guides you to the appropriate page for each field.

Table 8. Fields in MQGMO

This field...	Describes...	See page ...
GMSID	Structure identifier	27
GMVER	Structure version number	27
GMOPT	Options that control the action of MQGET	28
GMWI	Wait interval	46
GMSG1	Signal	47
GMRQN	Resolved name of destination queue	47
Note: The remaining fields are supported only in the version-2 structure.		
GMMO	Match options	47
GMGST	Group status	49
GMSST	Segment status	50
GMSEG	Segmentation	50

The current version of MQGMO is GMVER2. Fields that exist only in the version-2 structure are identified as such in the descriptions that follow. The declaration of MQGMO provided in the COPY file contains the new fields, but the initial value provided for the *GMVER* field is GMVER1; this ensures compatibility with existing applications. To use the new fields, the application must set the version number to GMVER2. Applications which are intended to be portable between several environments should use a version-2 MQGMO only if all of those environments support version 2.

The MQGMO structure is an input/output parameter for the MQGET call.

Fields

GMSID (4-byte character string)
Structure identifier.

The value must be:

GMSIDV
Identifier for get-message options structure.

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

GMVER (10-digit signed integer)
Structure version number.

The value must be one of the following:

GMVER1
Version-1 get-message options structure.

GMVER2
Version-2 get-message options structure.

Fields that exist only in the version-2 structure are identified as such in the descriptions that follow.

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

GMVERC
Current version of get-message options structure.

This is always an input field. The initial value of this field is GMVER1.

GMOPT (10-digit signed integer)
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). Combinations of options that are not valid are noted; all other combinations are valid. The following options are described. The page on which each description can be found is shown in the following table:

Option	See page
GMWT	28
GMNWT	29
GMSYP	29
GMPSYP	30
GMNSYP	30
GMBRWF	30
GMBRWN	32
GMBRWC	33
GMMUC	34
GMLK	35
GMUNLK	36
GMATM	36
GMFIQ	37
GMCONV	37
GMLOGO	38
GMCMPM	43
GMAMSA	45
GMASGA	45
GMNONE	46

GMWT
Wait for message to arrive.

The application is to wait until a suitable message arrives. The maximum time the application waits is specified in *GMWI*.

If MQGET requests are inhibited, or MQGET requests become inhibited while waiting, the wait is canceled and the call completes with CCFAIL and reason code RC2016, regardless of whether there are suitable messages on the queue.

This option can be used with the GMBRWF or GMBRWN 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* GMLK; an MQGET call specifying the GMLK option is treated as a *nonbrowse* call.

- If one or more nonbrowse MQGET calls is 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 shared 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 *MDMID* or *MDCID* (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 *MDMID* and *MDCID*.
- 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.

GMWT is ignored if specified with GMBRWC or GMMUC; no error is raised.

GMNWT
Return immediately if no suitable message.

The application is not to wait if no suitable message is available. This is the opposite of the GMWT option, and is defined to aid program documentation. It is the default if neither is specified.

GMSYP
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 GMNSYP is specified, the get request is not within a unit of work.

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

GMBRWF
GMBRWC
GMBRWN
GMLK
GMNSYP
GMPSYP
GMUNLK

GMPSYP
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 PEPER in the `MDPER` field in MQMD.

- If the message is persistent, the queue manager processes the call as though the application had specified GMSYP.
- If the message is not persistent, the queue manager processes the call as though the application had specified GMNSYP.

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

GMBRWF
GMBRWC
GMBRWN
GMCMPM
GMNSYP
GMSYP
GMUNLK

GMNSYP
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 a unit of work.

This option is assumed if GMBRWF or GMBRWN is specified.

If neither this option nor GMSYP is specified, the get request is not within a unit of work.

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

GMSYP
GMPSYP

GMBRWF
Browse from start of queue.

When a queue is opened with the OOBRW option, a browse cursor is established, positioned logically before the first message on the queue. Subsequent MQGET calls specifying the GMBRWF,

GMBRWN or GMBRWC 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 GMBRWN will search for a suitable message.

An MQGET call with GMBRWF 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 GMBRWN is issued, the browse cursor remains at the position in the queue that the message occupied, even though that position is now empty.

The GMMUC 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 CCFAIL, or a reason code of RC2080.

The GMLK option can be specified together with this option, to cause the message that is browsed to be locked.

GMBRWF can be specified with any valid combination of the GM* and MO*options that control the processing of messages in groups and segments of logical messages.

If GMLOGO is specified, the messages are browsed in logical order. If that option is omitted, the messages are browsed in physical order. When GMBRWF is specified, it is possible to switch between logical order and physical order, but subsequent MQGET calls using GMBRWN must browse the queue in the same order as the most-recent call that specified GMBRWF 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 GMBRWF 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 CCOK or CCWARN), 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:

GMBRWC
GMBRWN
GMMUC
GMSYP
GMPSYP
GMUNLK

It is also an error if the queue was not opened for browse.

GMBRWN
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 GMBRWF or GMBRWN option.

If the message is removed from the queue before the next MQGET call with GMBRWN 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 (MSPRIO), or
- FIFO *regardless* of priority (MSFIFO)

The *MsgDeliverySequence* queue attribute indicates which method applies (see page 253 for details).

If the queue has a *MsgDeliverySequence* of MSPRIO, 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 GMBRWN. It can only be found after the browse cursor has been reset with GMBRWF (or by reopening the queue).

The GMMUC 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 GMLK option can be specified together with this option, to cause the message that is browsed to be locked.

GMBRWN can be specified with any valid combination of the GM* and MO*options that control the processing of messages in groups and segments of logical messages.

If GMLOGO is specified, the messages are browsed in logical order. If that option is omitted, the messages are browsed in physical order. When GMBRWF is specified, it is possible to switch between logical order and physical order, but subsequent MQGET calls using GMBRWN must browse the queue in the same order as the most-recent call that specified GMBRWF for the queue handle. The call fails with reason code RC2259 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 GMLOGO 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 GMBRWN to browse beyond the end of the group, specifying MOSEQN with *MDSEQ* 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 MO* 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 GMBRWN 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:

GMBRWF
GMBRWC
GMMUC
GMSYP
GMPSYP
GMUNLK

It is also an error if the queue was not opened for browse.

GMBRWC
Browse message under browse cursor.

This option causes the message pointed to by the browse cursor to be retrieved nondestructively, regardless of the MO* options specified in the *GMMO* field in MQGMO.

The message pointed to by the browse cursor is the one that was last retrieved using either the GMBRWF or the GMBRWN 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 GMMUC 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 CCFAIL, or a reason code of RC2080.

If GMBRWC is specified *with* GMLK:

- 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 GMBRWC is specified *without* GMLK:

- 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 GMCMPM is specified with GMBRWC, the browse cursor must identify a message whose *MDOFF* field in MQMD is zero. If this condition is not satisfied, the call fails with reason code RC2246.

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:

GMBRWF
GMBRWN
GMMUC
GMSYP
GMPSYP
GMUNLK

It is also an error if the queue was not opened for browse.

GMMUC
Get message under browse cursor.

This option causes the message pointed to by the browse cursor to be retrieved, regardless of the MO* options specified in the *GMMO* 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 GMBRWF or the GMBRWN option.

If GMCMPM is specified with GMMUC, the browse cursor must identify a message whose *MDOFF* field in MQMD is zero. If this condition is not satisfied, the call fails with reason code RC2246.

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

GMBRWF
GMBRWC
GMBRWN
GMUNLK

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.

GMLK
: 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:

GMBRWF
GMBRWN
GMBRWC

Only one message can be locked per handle, but this can be a logical message or a physical message:

- If GMCMPM is specified, all of the message segments that constitute the logical message are locked to the queue handle (provided that they are all present on the queue and available for retrieval).
- If GMCMPM 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 GMCMPM 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 GMMUC option. Other MQGET calls for that queue handle can also remove the message (for example, a call that specifies the message identifier of the locked message).

If CCFAIL is returned (or CCWARN with RC2080), 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 GMBRWF or GMBRWN specified (with or without GMLK); the message is unlocked if the call completes with CCOK or CCWARN, but remains locked if the call completes with CCFAIL. However, the following exceptions apply:
  - The message *is not* unlocked if CCWARN is returned with RC2080.
  - The message *is* unlocked if CCFAIL is returned with RC2033.

  If GMLK is also specified, the new message is locked. If GMLK is not specified, there is no locked message after the call.

  If GMWT 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 GMBRWC (without GMLK); the message is unlocked if the call completes with CCOK or CCWARN, but remains locked if the call completes with CCFAIL. However, the following exception applies:
  - The message *is not* unlocked if CCWARN is returned with RC2080.
- Issuing another MQGET call for this handle with GMUNLK.
- 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 OOBRW, which is needed in order to specify the accompanying browse option.

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

GMSYP
GMPSYP
GMUNLK

GMUNLK
Unlock message.

The message to be unlocked must have been previously locked by an MQGET call with the GMLK option. If there is no message locked for this handle, the call completes with CCWARN and RC2209.

The *MSGDSC*, *BUFLEN*, *BUFFER*, and *DATLEN* parameters are not checked or altered if GMUNLK is specified. No message is returned in *BUFFER*.

No special open option is required to specify this option (although OOBRW is needed to issue the lock request in the first place).

This option is not valid with any options *except* the following:

GMNWT
GMNSYP

Both of these options are assumed whether specified or not.

GMATM
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 RC2079 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 RC2080 is returned if no other error occurs.

GMFIQ
Fail if queue manager is quiescing.

This option forces the MQGET call to fail if the queue manager is in the quiescing state.

If this option is specified together with GMWT, and the wait is outstanding at the time the queue manager enters the quiescing state:

- The wait is canceled and the call returns completion code CCFAIL with reason code RC2161.

If GMFIQ is not specified and the queue manager enters the quiescing state, the wait is not canceled.

GMCONV
Convert message data.

This option requests that the application data in the message should be converted, to conform to the *MDCSI* and *MDENC* values specified in the *MSGDSC* parameter on the MQGET call, before the data is copied to the *BUFFER* parameter.

The *MDFMT* 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 D, "Data-conversion" on page 385 for details of the data-conversion exit.

- If conversion is performed successfully, the *MDCSI* and *MDENC* fields specified in the *MSGDSC* 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 *MDCSI* and *MDENC* fields in *MSGDSC* are set to the values for the unconverted message. The completion code is CCWARN 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 description of the *MDFMT* field on page 78 for a list of format names for which the queue manager performs the conversion.

Group and segment options: The options described in the following text control the way that messages in groups and segments of logical messages are returned by the MQGET call. The following definitions may be of help in understanding these 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 (*MDMID* 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 (*MDGID* field in MQMD), and the same message sequence number (*MDSEQ* field in MQMD). The segments are distinguished by differing values for the segment offset (*MDOFF* 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 (GINONE), 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.

GMLOGO

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 GMLOGO 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 GMLOGO 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 GMLOGO 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 *MDGID*, *MDSEQ*, and *MDOFF* fields in MQMD. However, the application does need to set the GMSYP or GMNSYP 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 MFMIG flag is returned by the MQGET call. With GMLOGO specified on successive calls, that group remains the current group until a message is returned that has:

- MFLMIG without MFSEG (that is, the last logical message in the group is not segmented), or
- MFLMIG with MFLSEG (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 MFSEG flag is returned by the MQGET call, and that logical message is terminated when the message that has the MFLSEG 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 *GMMO* field:

MOMSGI
MOCORI
MOGRPI

However, these options are effective only when there is no current message group or logical message; see *GMMO* on page 47 for further details.

Table 9 on page 40 shows the values of the *MDMID*, *MDCID*, *MDGID*, *MDSEQ*, and *MDOFF* 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 abbreviated column headings have the following meanings:

- **LOG ORD** means the GMLOGO option.
- **Cur grp** means that a current message group exists prior to the call.
- **Cur log msg** means that a current logical message exists prior to the call.

In the table:

- "(√)" indicates that the row applies whether or not there is a √ in that column.
- "Previous" denotes the value returned for that field in the previous message for the queue handle.

Table 9. 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**	*MDMID*	*MDCID*	*MDGID*	*MDSEQ*	*MDOFF*
√			Controlled by *GMMO*	Controlled by *GMMO*	Controlled by *GMMO*	1	0
√		√	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 *GMMO*	Controlled by *GMMO*	Controlled by *GMMO*	Controlled by *GMMO*	Controlled by *GMMO*

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 GMLOGO 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 RC2245.

When GMLOGO is specified, the MQGMO supplied on the MQGET call must not be less than GMVER2, and the MQMD must not be less than MDVER2. If this condition is not satisfied, the call fails with reason code RC2256 or RC2257, as appropriate.

If GMLOGO 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 MO*options specified on those calls (see *GMMO* on page 47 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 *MDGID*, *MDSEQ*, *MDOFF*, and *GMMO* fields to the appropriate values, and then issue the MQGET call with GMSYP or GMNSYP set as desired, but *without* specifying GMLOGO. If this call is successful, the queue manager retains the group and segment information, and subsequent MQGET calls using that queue handle can specify GMLOGO 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 GMLOGO with MQGET calls that do not, but the following points should be noted:

- Each successful MQGET call that does *not* specify GMLOGO 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 GMLOGO 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 CCWARN completion code. Table 10 shows the various cases that can arise. In these cases, if the completion code is not CCOK, the reason code is one of the following (as appropriate):

  RC2241
  RC2242
  RC2245

  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 CCOK (assuming no other errors).

Table 10. Outcome when MQGET or MQCLOSE call not consistent with group and segment information

Current call	**Previous call**	
	MQGET with GMLOGO	MQGET without GMLOGO
MQGET with GMLOGO	CCFAIL	CCFAIL
MQGET without GMLOGO	CCWARN	CCOK
MQCLOSE with an unterminated group or logical message	CCWARN	CCOK

Applications that simply want to retrieve messages and segments in logical order are recommended to specify GMLOGO, 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 GMLOGO option, and this can be achieved by not specifying that option. If this is done, the application must ensure that the *MDMID*, *MDCID*, *MDGID*, *MDSEQ*, and *MDOFF* fields in MQMD, and the MO*options in *GMMO* 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 GMLOGO. 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 GMLOGO, nor the corresponding PMLOGO 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.

GMLOGO can be specified with any of the other GM* options, and with various of the MO* options in appropriate circumstances (see above).

GMCMPM

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 GMCMPM.

To use this option, the application must provide a buffer which is big enough to accommodate the complete message, or specify the GMATM 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 GMCMPM 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 RC2255.

For *nonpersistent* messages, the queue manager does not require a unit of work to be available in order to perform reassembly.

Each physical message which 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 *MDMID*, *MDOFF*, and *MDMFL* fields that differ between segments in the logical message. However, when segments take different paths through the network, and some of those paths have MCA sender conversion enabled, it is possible for the *MDCSI* and *MDENC* fields to differ between segments when the segments eventually arrive at the destination queue. A logical message consisting of segments in which the *MDCSI* and/or *MDENC* 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 CCWARN and reason code RC2243 or RC2244, as appropriate. This happens regardless of whether GMCONV is specified. To retrieve the remaining segments, the application must reissue the MQGET call without the GMCMPM option, retrieving the segments one by one. GMLOGO 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 GMCMPM 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 *MDMFL* field, which the queue manager sets to indicate that the reassembled message is the only segment.

If GMCMPM 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 GMCMPM. 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 RO*D or RO*F 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 GMCMPM is specified with GMMUC or GMBRWC, the browse cursor must be positioned on a message whose *MDOFF* field in MQMD has a value of 0. If this condition is not satisfied, the call fails with reason code RC2246.

GMCMPM implies GMASGA, which need not therefore be specified.

GMCMPM can be specified with any of the other GM* options apart from GMPSYP, and with any of the MO* options apart from MOOFFS.

GMAMSA
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 GMAMSA 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 RC2033 is returned after the specified wait interval (if any) has expired.

The processing of GMAMSA depends on whether GMLOGO is also specified:

- If both options are specified, GMAMSA has an effect *only* when there is no current group or logical message. If there *is* a current group or logical message, GMAMSA is ignored. This means that GMAMSA can remain on when processing messages in logical order.
- If GMAMSA is specified without GMLOGO, GMAMSA *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.

GMAMSA implies GMASGA, which need not therefore be specified.

GMAMSA can be specified with any of the other GM* options, and with any of the MO* options.

GMASGA
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 GMASGA 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 RC2033 is returned after the specified wait interval (if any) has expired.

The processing of GMASGA depends on whether GMLOGO is also specified:

- If both options are specified, GMASGA has an effect *only* when there is no current logical message. If there *is* a current logical message, GMASGA is ignored. This means that GMASGA can remain on when processing messages in logical order.

- If GMASGA is specified without GMLOGO, GMASGA *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 GMCMPM and GMASGA 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 GMASGA 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 GMASGA 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 GMASGA does not imply that GMCMPM 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.

GMASGA can be specified with any of the other GM* options, and with any of the MO* options.

GMNONE
: No options specified.

This value can be used to indicate that no other options have been specified; all options assume their default values. GMNONE 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 GMNWT.

GMWI (10-digit signed integer)
: 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 *MSGDSC* parameter of the MQGET call; see the description of the *MDMID* field on page 83 for more details). If no suitable message has arrived after this time has elapsed, the call completes with CCFAIL and reason code RC2033.

GMWI is used in conjunction with the GMWT option. It is ignored if this option is not specified. If it is specified, *GMWI* must be greater than or equal to zero, or the following special value:

WIULIM
: An unlimited wait is required.

The initial value of this field is 0.

GMSG1 (10-digit signed integer)
Signal.

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

GMSG2 (10-digit signed integer)
Signal identifier.

This is a reserved field; its value is not significant.

GMRQN (48-byte character string)
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 LNQN. The initial value of this field is 48 blank characters.

The remaining fields in this structure are not present if *GMVER* is less than GMVER2.

GMMO (10-digit signed integer)
Options controlling selection criteria used for MQGET.

These options allow the application to choose which fields in the *MSGDSC* 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 *MSGDSC* 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 *MSGDSC* 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), *GMMO* should be set to MONONE.

If GMLOGO 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 *MDSEQ* equal to 1 and *MDOFF* 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:

  MOMSGI
  MOCORI
  MOGRPI

- 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 MO* 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 *MSGDSC* parameter must match the value of the corresponding field in the message to be returned; the call fails with reason code RC2247 is this condition is not satisfied.

GMMO is ignored if either GMMUC or GMBRWC is specified.

One or more of the following match options can be specified:

MOMSGI
: 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 *MDMID* field in the *MSGDSC* 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 *MDMID* field in the *MSGDSC* parameter is ignored, and any message identifier will match.

  Note: The message identifier MINONE is a special value that matches *any* message identifier in the MQMD for the message. Therefore, specifying MOMSGI with MINONE is the same as *not* specifying MOMSGI.

MOCORI
: 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 *MDCID* field in the *MSGDSC* 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 *MDCID* field in the *MSGDSC* parameter is ignored, and any correlation identifier will match.

  Note: The correlation identifier CINONE is a special value that matches *any* correlation identifier in the MQMD for the message. Therefore, specifying MOCORI with CINONE is the same as *not* specifying MOCORI.

MOGRPI
: 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 *MDGID* field in the *MSGDSC* 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 *MDGID* field in the *MSGDSC* parameter is ignored, and any group identifier will match.

  Note: The group identifier GINONE is a special value that matches *any* group identifier in the MQMD for the message. Therefore, specifying MOGRPI with GINONE is the same as *not* specifying MOGRPI.

MOSEQN
: 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 *MDSEQ* field in the *MSGDSC* 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 *MDSEQ* field in the *MSGDSC* parameter is ignored, and any message sequence number will match.

MOOFFS
: Retrieve message with specified offset.

  This option specifies that the message to be retrieved must have an offset that matches the value of the *MDOFF* field in the *MSGDSC* 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 *MDOFF* field in the *MSGDSC* parameter is ignored, and any offset will match.

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

MONONE
: 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 GMAMSA, GMASGA, and GMCMPM options).

  MONONE 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 this field is MOMSGI with MOCORI. This field is not present if *GMVER* is less than GMVER2.

Note: The initial value of the *GMMO* 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 *MDMID* and *MDCID* fields to MINONE and CINONE prior to each MQGET call. The need to reset *MDMID* and *MDCID* can be avoided by setting *GMVER* to GMVER2, and *GMMO* to MONONE.

GMGST (1-byte character string)
: Flag indicating whether message retrieved is in a group.

  It has one of the following values:

  GSNIG
  : Message is not in a group.

  GSMIG
  : Message is in a group, but is not the last in the group.

GSLMIG
: Message is the last in the group.

  This is also the value returned if the group consists of only one message.

This is an output field. The initial value of this field is GSNIG. This field is not present if *GMVER* is less than GMVER2.

GMSST (1-byte character string)
: Flag indicating whether message retrieved is a segment of a logical message.

  It has one of the following values:

  SSNSEG
  : Message is not a segment.

  SSSEG
  : Message is a segment, but is not the last segment of the logical message.

  SSLSEG
  : Message is the last segment of the logical message.

    This is also the value returned if the logical message consists of only one segment.

  This is an output field. The initial value of this field is SSNSEG. This field is not present if *GMVER* is less than GMVER2.

GMSEG (1-byte character string)
: Flag indicating whether further segmentation is allowed for the message retrieved.

  It has one of the following values:

  SEGIHB
  : Segmentation not allowed.

  SEGALW
  : Segmentation allowed.

  This is an output field. The initial value of this field is SEGIHB. This field is not present if *GMVER* is less than GMVER2.

GMRE1 (1-byte character string)
: Reserved.

  This is a reserved field. The initial value of this field is a blank character. This field is not present if *GMVER* is less than GMVER2.

Table 11. Initial values of fields in MQGMO

Field name	Name of constant	Value of constant
GMSID	GMSIDV	'GMOb' (See note 1)
GMVER	GMVER1	1
GMOPT	GMNWT	0
GMWI	None	0
GMSG1	None	0
GMSG2	None	0
GMRQN	None	Blanks
GMMO	MOMSGI + MOCORI	3
GMGST	GSNIG	'b'
GMSST	SSNSEG	'b'
GMSEG	SEGIHB	'b'
GMRE1	None	'b'

Notes:

1. The symbol ‘b’ represents a single blank character.

RPG declaration (ILE)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQGMO Structure
D*
D* Structure identifier
D  GMSID                  1      4
D* Structure version number
D  GMVER                  5      8I 0
D* Options that control the action of MQGET
D  GMOPT                  9     12I 0
D* Wait interval
D  GMWI                  13     16I 0
D* Signal
D  GMSG1                 17     20I 0
D* Signal identifier
D  GMSG2                 21     24I 0
D* Resolved name of destination queue
D  GMRQN                 25     72
D* Options controlling selection criteria used for MQGET
D  GMMO                  73     76I 0
D* Flag indicating whether message retrieved is in a group
D  GMGST                 77     77
D* Flag indicating whether message retrieved is a segment of a
D* logical message
D  GMSST                 78     78
D* Flag indicating whether segmentation is allowed for the message
D* retrieved
D  GMSEG                 79     79
D* Reserved
D  GMRE1                 80     80
```

RPG declaration (OPM)

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7..
I* MQGMO Structure
I*
I* Structure identifier
I                                       1   4 GMSID
I* Structure version number
I                                    B   5   80GMVER
I* Options that control the action of MQGET
I                                    B   9  120GMOPT
I* Wait interval
I                                    B  13  160GMWI
I* Signal
I                                    B  17  200GMSG1
I* Signal identifier
I                                    B  21  240GMSG2
I* Resolved name of destination queue
I                                       25  72 GMRQN
I* Options controlling selection criteria used for MQGET
I                                    B  73  760GMMO
I* Flag indicating whether message retrieved is in a group
I                                       77  77 GMGST
I* Flag indicating whether message retrieved is a segment of a
I* logical message
I                                       78  78 GMSST
I* Flag indicating whether further segmentation is allowed for the
I* message retrieved
I                                       79  79 GMSEG
I* Reserved
I                                       80  80 GMRE1
```

MQIIH – IMS header

The following table guides you to the appropriate page for each field.

Table 12. Fields in MQIIH

This field...	Describes...	See page ...
IISID	Structure identifier	54
IIVER	Structure version number	54
IILEN	Length of MQIIH structure	54
IIFMT	Format name	54
IILTO	Logical terminal override	55
IIMMN	Message format services map name	55
IIRFM	Format name of reply message	55
IIAUT	RACF password or passticket	55
IITID	Transaction instance identifier	55
IITST	Transaction state	56
IICMT	Commit mode	56
IISEC	Security scope	56

The MQIIH structure describes the information that must be present at the start of a message sent to the IMS bridge through MQSeries for MVS/ESA. The format name of this structure is FMIMS.

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 which 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 and application message data is performed by the queue manager as necessary.

  Note: There is one exception to this. If the queue manager which owns the IMS bridge queue is using CICS for distributed queuing, the MQIIH must be in the character set and encoding of that queue manager.
- The application message data following the MQIIH structure must be in the same character set and encoding as the MQIIH structure. The *IICSI* and *IIENC* fields in the MQIIH structure cannot be used to specify the character set and encoding of the application message data.

Fields

IISID (4-byte character string)
Structure identifier.

The value must be:

IISIDV
Identifier for IMS information header structure.

The initial value of this field is IISIDV.

IIVER (10-digit signed integer)
Structure version number.

The value must be:

IIVER1
Version number for IMS information header structure.

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

IIVERC
Current version of IMS information header structure.

The initial value of this field is IIVER1.

IILEN (10-digit signed integer)
Length of MQIIH structure.

The value must be:

IILEN1
Length of IMS information header structure.

The initial value of this field is IILEN1.

IIENC (10-digit signed integer)
Reserved.

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

IICSI (10-digit signed integer)
Reserved.

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

IIFMT (8-byte character string)
MQ format name.

This is the MQ format name of the application message data which follows the MQIIH structure. The rules for coding this are the same as those for the *MDFMT* field in MQMD.

The length of this field is given by LNFMT. The initial value of this field is FMNONE.

IIFLG (10-digit signed integer)
Reserved.

The value must be:

IINONE
No flags.

The initial value of this field is IINONE.

IILTO (8-byte character string)
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 LNLTOV. The initial value of this field is 8 blank characters.

IIMMN (8-byte character string)
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 LNMFMN. The initial value of this field is 8 blank characters.

IIRFM (8-byte character string)
MQ format name of reply message.

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

The length of this field is given by LNFMT. The initial value of this field is FMNONE.

IIAUT (8-byte character string)
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:

IAUNON
No authentication.

The length of this field is given by LNAUTH. The initial value of this field is IAUNON.

IITID (16-byte bit string)
Transaction instance identifier.

This field is used by output messages from IMS so is ignored on first input. If *IITST* is set to ITSIC, 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:

ITINON
No transaction instance id.

The length of this field is given by LNTIID. The initial value of this field is ITINON.

IITST (1-byte character string)
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:

ITSIC
In conversation.

ITSNIC
Not in conversation.

The initial value of this field is ITSNIC.

IICMT (1-byte character string)
Commit mode.

See the *OTMA User's Guide* for more information about IMS commit modes. The value must be one of the following:

ICMCTS
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.

ICMSTC
Send then commit.

The initial value of this field is ICMCTS.

IISEC (1-byte character string)
Security scope.

This indicates the desired IMS security processing. The value must be one of the following:

ISSCHK
Check security scope.

An ACEE is built in the control region, but not in the dependent region.

ISSFUL
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 ISSFUL, you must ensure that the user ID for which the ACEE is built has access to the resources used in the dependent region.

The initial value of this field is ISSCHK.

IIRSV (1-byte character string)
Reserved.

This is a reserved field; it must be blank.

Table 13. Initial values of fields in MQIIH

Field name	Name of constant	Value of constant
IISID	IISIDV	'IIHb' (See note 1)
IIVER	IIVER1	1
IILEN	IILEN1	84
IIENC	None	0
IICSI	None	0
IIFMT	FMNONE	'bbbbbbbb'
IIFLG	IINONE	0
IILTO	None	'bbbbbbbb'
IIMMN	None	'bbbbbbbb'
IIRFM	FMNONE	'bbbbbbbb'
IIAUT	IAUNON	'bbbbbbbb'
IITID	ITINON	Nulls
IITST	ITSNIC	'b'
IICMT	ICMCTS	'0'
IISEC	ISSCHK	'C'
IIRSV	None	'b'

Notes:

1. The symbol 'b' represents a single blank character.

RPG declaration (ILE)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQIIH Structure
D*
D* Structure identifier
D  IISID                 1      4
D* Structure version number
D  IIVER                 5      8I 0
D* Length of MQIIH structure
D  IILEN                 9     12I 0
D* Reserved
D  IIENC                13     16I 0
D* Reserved
D  IICSI                17     20I 0
D* MQ format name
D  IIFMT                21     28
D* Reserved
D  IIFLG                29     32I 0
D* Logical terminal override
D  IILTO                33     40
D* Message format services map name
D  IIMMN                41     48
D* MQ format name of reply message
D  IIRFM                49     56
D* RACF password or passticket
D  IIAUT                57     64
```

```
D* Transaction instance identifier
D  IITID                 65     80
D* Transaction state
D  IITST                 81     81
D* Commit mode
D  IICMT                 82     82
D* Security scope
D  IISEC                 83     83
D* Reserved
D  IIRSV                 84     84
```

RPG declaration (OPM)

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7..
I* MQIIH Structure
I*
I* Structure identifier
I                                       1   4 IISID
I* Structure version number
I                                    B   5   80IIVER
I* Length of MQIIH structure
I                                    B   9  120IILEN
I* Reserved
I                                    B  13  160IIENC
I* Reserved
I                                    B  17  200IICSI
I* MQ format name
I                                      21  28 IIFMT
I* Reserved
I                                    B  29  320IIFLG
I* Logical terminal override
I                                      33  40 IILTO
I* Message format services map name
I                                      41  48 IIMMN
I* MQ format name of reply message
I                                      49  56 IIRFM
I* RACF password or passticket
I                                      57  64 IIAUT
I* Transaction instance identifier
I                                      65  80 IITID
I* Transaction state
I                                      81  81 IITST
I* Commit mode
I                                      82  82 IICMT
I* Security scope
I                                      83  83 IISEC
I* Reserved
I                                      84  84 IIRSV
```

MQMD – Message descriptor

The following table guides you to the appropriate page for each field.

Table 14. Fields in MQMD

This field...	Describes...	See page ...
MDSID	Structure identifier	61
MDVER	Structure version number	61
MDREP	Options for report messages	61
MDMT	The type of message	71
MDEXP	Message lifetime	72
MDFB	Feedback code	74
MDENC	Data encoding	77
MDCSI	Coded character set identifier	77
MDFMT	Format name	78
MDPRI	Message priority	81
MDPER	Message persistence	82
MDMID	Message identifier	83
MDCID	Correlation identifier	85
MDBOC	Backout counter	85
MDRQ	Name of reply queue	85
MDRM	Name of reply queue manager	86
MDUID	User identifier	87
MDACC	Accounting token	87
MDAID	Application data relating to identity	88
MDPAT	Type of application that put the message	89
MDPAN	Identity of application that put the message	90
MDPD	Date when message was put	90
MDPT	Time when message was put	91
MDAOD	Application data relating to origin	92
Note: The remaining fields are supported only in the version-2 structure.		
MDGID	Group identifier	92
MDSEQ	Message sequence number	94
MDOFF	Offset	94
MDMFL	Message flags	95
MDOLN	Original length	99

The MQMD structure contains the control information that accompanies the application data when a message travels between the sending and receiving applications.

Character data in the message descriptor is in the character set of the queue manager to which the application is connected; this is given by the *CodedCharSetId* queue-manager attribute. Numeric data in the message descriptor is in the native machine encoding (given by ENNAT).

If the sending and receiving queue managers use different character sets or encodings, the data in the message descriptor is converted automatically—it is not necessary for the receiving application to perform these conversions.

If the application message data requires conversion, this can be accomplished by means of a user-written exit invoked when the message is retrieved using the MQGET call. For further information, see:

- The description of the GMCONV option on page 37
- Usage note describing GMCONV under the MQGET call
- *MQSeries Application Programming Guide*

When a message is on a transmission queue, some of the fields in MQMD are set to particular values; see page 159 for details.

The current version of MQMD is MDVER2. Fields that exist only in the version-2 structure are identified as such in the descriptions that follow. The declaration of MQMD provided in the COPY file contains the new fields, but the initial value provided for the *MDVER* field is MDVER1; this ensures compatibility with existing applications. To use the new fields, the application must set the version number to MDVER2. A declaration for the version-1 structure is available with the name MQMD1. Applications which are intended to be portable between several environments should use a version-2 MQMD only if all of those environments support version 2.

A version-2 MQMD is generally equivalent to using a version-1 MQMD and prefixing the application 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 *MDFMT* field in MQMD to FMMDE 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 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.

- 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 *MDFMT* field in MQMD will have the value FMMDE 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 18 on page 108.

This structure is an input/output parameter for the MQGET, MQPUT, and MQPUT1 calls.

Fields

MDSID (4-byte character string)
Structure identifier.

The value must be:

MDSIDV
Identifier for message descriptor structure.

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

MDVER (10-digit signed integer)
Structure version number.

The value must be one of the following:

MDVER1
Version-1 message descriptor structure.

MDVER2
Version-2 message descriptor structure.

Fields that exist only in the version-2 structure are identified as such in the descriptions that follow.

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

MDVERC
Current version of message descriptor structure.

This is always an input field. The initial value of this field is MDVER1.

MDREP (10-digit signed integer)
Report options.

A report is a message about another message, used to inform an application about expected or unexpected events that relate to the original message. The *MDREP* 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 report types 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).

The application that receives the report message can determine the reason the report was generated by examining the *MDFB* field in the MQMD. See page 74 for more details.

Exception options: You can specify one of the following to request an exception report message:

ROEXC
: 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.

  The exception report is generated only if the original message can be placed successfully on the dead-letter queue (if the RODLQ action was specified), or discarded (if the RODISC action was specified). If the action specified cannot be completed successfully, the original message is left on the transmission queue, and no exception report message is generated.

  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 ROEXC, ROEXCD, and ROEXCF.

ROEXCD
: Exception reports with data required.

  This is the same as ROEXC, 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 ROEXC, ROEXCD, and ROEXCF.

ROEXCF
: Exception reports with full data required.

  This is the same as ROEXC, 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 ROEXC, ROEXCD, and ROEXCF.

Expiration options: You can specify one of the following to request an expiration report message:

ROEXP
: 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 *MDEXP* 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 ROEXC* options is specified).

Message data from the original message is not included with the report message.

Do not specify more than one of ROEXP, ROEXPD, and ROEXPF.

ROEXPD
: Expiration reports with data required.

This is the same as ROEXP, 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 ROEXP, ROEXPD, and ROEXPF.

ROEXPF
: Expiration reports with full data required.

This is the same as ROEXP, 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 ROEXP, ROEXPD, and ROEXPF.

Confirm-on-arrival options: You can specify one of the following to request a confirm-on-arrival report message:

ROCOA
: 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 *MDFMT* field in the message descriptor is FMXQH or FMDLH. 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 ROCOA, ROCOAD, and ROCOAF.

ROCOAD
: Confirm-on-arrival reports with data required.

This is the same as ROCOA, 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 ROCOA, ROCOAD, and ROCOAF.

ROCOAF
: Confirm-on-arrival reports with full data required.

This is the same as ROCOA, 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 ROCOA, ROCOAD, and ROCOAF.

Confirm-on-delivery options: You can specify one of the following to request a confirm-on-delivery report message:

ROCOD
: 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 generated if the *MDFMT* field in the message descriptor is FMDLH. This prevents a COD report being generated if the message is undeliverable and put on a dead-letter queue.

ROCOD is not valid if the destination queue is an XCF queue.

Do not specify more than one of ROCOD, ROCODD, and ROCODF.

ROCODD
: Confirm-on-delivery reports with data required.

This is the same as ROCOD, 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.

Note that any truncation of the original message on retrieval (using the GMATM option) has no effect on the size of the message data in the COD report.

ROCODD is not valid if the destination queue is an XCF queue.

Do not specify more than one of ROCOD, ROCODD, and ROCODF.

ROCODF
: Confirm-on-delivery reports with full data required.

This is the same as ROCOD, except that all of the application message data from the original message is included in the report message.

ROCODF is not valid if the destination queue is an XCF queue.

Do not specify more than one of ROCOD, ROCODD, and ROCODF.

Action-notification options: You can specify one or both of the following to request that the receiving application send a positive-action or negative-action report message:

ROPAN
: 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.

RONAN
: 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 following to control how the *MDMID* of the report message (or of the reply message) is to be set:

RONMI
: 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 *MDMID* is to be generated for the report or reply message.

ROPMI
: Pass message identifier.

  If a report or reply is generated as a result of this message, the

MDMID of this message is to be copied to the *MDMID* of the report or reply message.

If this option is not specified, RONMI is assumed.

Correlation-identifier options: You can specify one of the following to control how the *MDCID* of the report message (or of the reply message) is to be set:

ROCMTC
: 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 *MDMID* of this message is to be copied to the *MDCID* of the report or reply message.

ROPCI
: Pass correlation identifier.

  If a report or reply is generated as a result of this message, the *MDCID* of this message is to be copied to the *MDCID* of the report or reply message.

  If this option is not specified, ROCMTC is assumed.

Servers replying to requests or generating report messages are recommended to check whether the ROPMI or ROPCI 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 following to control the disposition of the original message when it cannot be delivered to the destination queue:

RODLQ
: 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.

RODISC
: Discard message.

  This indicates that the message should be discarded if it cannot be delivered to the destination queue. An exception report message will be generated, if one was requested by the sender.

  If it is desired to return the original message to the sender, without the original message being placed on the dead-letter queue, the sender should specify RODISC with ROEXCF.

Default option: You can specify the following if no report options are required:

RONONE
: No reports required.

  This value can be used to indicate that no other options have been specified. RONONE 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 (with or without the data option in either case), 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 *MDREP* options are set, no report messages are generated by the queue manager or message channel agent.

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 C, “Report options” on page 379 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 *MDRQ* field. When a report message is received, the nature of the report can be determined by examining the *MDFB* field in the message descriptor.

If the queue manager or message channel agent 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 *MDRQ* queue is opened to put the report using the authority of the *MDUID* of the original message, except in the following cases:

- Exception reports generated by a receiving message channel agent use the same authority as was used to put the original message.
- COA reports generated by the queue manager use the same authority as was used to put the original message.

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 ID 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 *MDFMT* of the original message is FMXQH, 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 a message channel agent 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 MFSEGA flag on page 95). 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 *MDGID* field in the report message can be used to correlate the multiple reports with the group identifier of the original message, and the *MDFB* field used to identify the type of each report message.

If GMLOGO 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 GMCMPM option (optionally with GMATM). GMCMPM 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 GMCMPM option, only the report options in the *first* segment will be honored by the queue manager.
- If the segments are retrieved one by one, and they do not all have the same setting of the COD report option, it may not be possible to use the GMCMPM option to retrieve the report messages with a single MQGET call, or the GMASGA 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 RO*D or RO*F options. However, be aware that if RO*D 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 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
MDSID	MDSIDV
MDVER	MDVER1
MDREP	RONONE
MDMT	MTRPRT
MDEXP	EIULIM
MDFB	As appropriate for the nature of the report (FBCOA, FBCOD, FBEXP, or an RC* value)
MDENC	Copied from the original message descriptor
MDCSI	Copied from the original message descriptor
MDFMT	Copied from the original message descriptor
MDPRI	Copied from the original message descriptor
MDPER	Copied from the original message descriptor
MDMID	As specified by the report options in the original message descriptor
MDCID	As specified by the report options in the original message descriptor
MDBOC	0
MDRQ	Blanks
MDRM	Name of queue manager
MDUID	As set by the PMPASI option
MDACC	As set by the PMPASI option
MDAID	As set by the PMPASI option
MDPAT	ATQM, or as appropriate for the message channel agent
MDPAN	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.
MDPD	Date when report message is sent
MDPT	Time when report message is sent
MDAOD	Blanks
MDGID	Copied from the original message descriptor
MDSEQ	Copied from the original message descriptor
MDOFF	Copied from the original message descriptor
MDMFL	Copied from the original message descriptor
MDOLN	Copied from the original message descriptor if not OLUNDF, 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 *MDRM* 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 PMPASI.

Analyzing the report field: The *MDREP* 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 381.

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 RONONE.

MDMT (10-digit signed integer)
Message type.

This indicates the type of the message. Message types are grouped as follows:

MTSFST
Lowest value for system-defined message types.

MTSLST
Highest value for system-defined message types.

The following values are currently defined within the system range:

MTDGRM
Message not requiring a reply.

The message is one that does not require a reply.

MTRQST
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 *MDRQ* field. The *MDREP* field indicates how the *MDMID* and *MDCID* of the reply are to be set.

MTRPLY
Reply to an earlier request message.

The message is the reply to an earlier request message (MTRQST). The message should be sent to the queue indicated by the *MDRQ* field of the request message. The *MDREP* field of the request should be used to control how the *MDMID* and *MDCID* of the reply are set.

Note: The queue manager does not enforce the request-reply relationship; this is an application responsibility.

MTRPRT
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 *MDRQ* field of the message descriptor of the original message. The *MDFB* field should be set to indicate the nature of the report. The *MDREP* field of the original message can be used to control how the *MDMID* and *MDCID* of the report message should be set.

Report messages generated by the queue manager or message channel agent are always sent to the *MDRQ* queue, with the *MDFB* and *MDCID* 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:

MTAFST
: Lowest value for application-defined message types.

MTALST
: Highest value for application-defined message types.

For the MQPUT and MQPUT1 calls, the *MDMT* value must be within either the system-defined range or the application-defined range; if it is not, the call fails with reason code RC2029.

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 MTDGRM.

MDEXP (10-digit signed integer)
: Expiry time.

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 *MDEXP* 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. It is not defined, however, precisely when a message that is eligible for discarding is actually discarded; the discard may not occur until a non-browse MQGET call is issued that would have retrieved the message, or even later. However, 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 *MDEXP* field of the message descriptor after a successful MQGET call is either greater than zero, or the special value EIULIM.

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 ROEXP* 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 *MDEXP* time of zero, the MQPUT or MQPUT1 call fails with reason code RC2013; 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. Similarly, if a nearly-expired message is locked by an MQGET call with GMLK, the message may become eligible to be discarded before it can be retrieved by an MQGET call with GMMUC; reason code RC2034 is returned on this subsequent MQGET call if that happens.
7. Servers should not normally reflect the unused expiry time of a request in the reply; the default action should be to put the reply with EIULIM. However, the default action for putting messages to a dead-letter (undelivered-message) queue is to preserve the outstanding expiry time of the message, and to continue to decrement it.
8. Trigger messages are always generated with EIULIM.
9. A message (normally on a transmission queue) which has a *MDFMT* name of FMXQH has a second message descriptor within the MQXQH. It therefore has two *MDEXP* 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 *MDEXP* 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 FMXQH (but the queue manager does not enforce this). In this case the application need not set the values of these two *MDEXP* fields to be the same. (The queue manager does not check that the *MDEXP* 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 *MDFMT* name of FMXQH is retrieved from a queue (whether this is a normal or a transmission queue), the queue manager decrements *both* these *MDEXP* fields with the time spent waiting on the queue. No error is raised if the message data is not long enough to include the *MDEXP* field in the MQXQH.
- The queue manager uses the *MDEXP* 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 *MDEXP* fields were different, it is therefore possible for the *MDEXP* 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 *MDEXP* field in the MQXQH has elapsed. In this case the *MDEXP* field in the MQXQH is set to zero.

The following special value is recognized:

EIULIM
: Unlimited lifetime.

  The message has an unlimited expiration time.

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 EIULIM.

MDFB (10-digit signed integer)
: Feedback or reason code.

  This is used with a message of type MTRPRT to indicate the nature of the report, and is only meaningful with that type of message.

  Feedback codes are grouped as follows:

  FBNONE
  : No feedback provided.

  FBSFST
  : Lowest value for system-generated feedback.

  FBSLST
  : Highest value for system-generated feedback.

    The range of system-generated feedback codes FBSFST through FBSLST includes the special feedback codes (FB*) listed below, and also the reason codes (RC*) that can occur when the message cannot be put on the destination queue.

  FBAFST
  : Lowest value for application-generated feedback.

  FBALST
  : Highest value for application-generated feedback.

  Applications that generate report messages should not use feedback codes in the system range (other than FBQUIT), 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 be within either the system range or the application range. This is checked whatever the value of *MDMT*.

Special feedback codes are:

FBEXP
: Message expired.

  Message was discarded because it had not been removed from the destination queue before its expiry time had elapsed.

FBCOA
: Confirmation of arrival on the destination queue (see ROCOA).

FBCOD
: Confirmation of delivery to the receiving application (see ROCOD).

FBPAN
: Positive action notification (see ROPAN).

FBNAN
: Negative action notification (see RONAN).

FBQUIT
: 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 MTRPRT 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.

FBDLZ
: Data length zero.

  A segment length was zero in the application data of the message.

FBDLN
: Data length negative.

  A segment length was negative in the application data of the message.

FBDLTB
: Data length too big.

  A segment length was too big in the application data of the message.

FBBUFO
: Buffer overflow.

  The value of one of the length fields would cause the data to overflow the MQSeries message buffer.

FBLOB1
: Length in error by one.

  The value of one of the length fields was one byte too short.

FBIIH
MQIIH structure not valid or missing.

The *MDFMT* field in MQMD specifies FMIMS, but the message does not begin with a valid MQIIH structure.

FBNAFI
Userid not authorized for use in IMS.

The user ID contained in the message descriptor MQMD, or the password contained in the *IIAUT* field in the MQIIH structure, failed the validation performed by the IMS bridge. As a result the message was not passed to IMS.

FBIERR
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.

FBIFST
Lowest value for IMS-generated feedback.

IMS-generated feedback codes occupy the range FBIFST through FBILST. The IMS error code itself is *MDFB* minus FBIERR.

FBILST
Highest value for IMS-generated feedback.

Reason codes are used for exception reports. They include:

RC2051
(2051, X'803') Put calls inhibited for the queue.

RC2053
(2053, X'805') Queue already contains maximum number of messages.

RC2035
(2035, X'7F3') Not authorized for access.

RC2056
(2056, X'808') No space available on disk for queue.

RC2048
(2048, X'800') Message on a temporary dynamic queue cannot be persistent.

RC2031
(2031, X'7EF') Message length greater than maximum for queue manager.

RC2030
(2030, X'7EE') Message length greater than maximum for queue.

For a full list of reason codes, see "Reason code" on page 275.

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 FBNONE.

MDENC (10-digit signed integer)
Data encoding.

This identifies the representation used for numeric values in the application message data; this applies to binary integer data, packed-decimal integer data, and floating-point data. The following value is defined:

ENNAT
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 is programming-language and environment specific.

The queue manager does not validate the contents of this field.

Applications that put messages should normally specify ENNAT. Applications that retrieve messages should compare this field against the value ENNAT; if the values differ, the application may need to convert numeric data in the message. See Appendix B, “Machine encodings” on page 375 for details of how this field is constructed.

If the GMCONV 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.

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 ENNAT.

MDCSI (10-digit signed integer)
Coded character set identifier.

This specifies the coded character set identifier of character data in the application message data.

Note that character data in the message descriptor and the other MQI data structures must be in the character set used by the queue manager. This is defined by the queue manager’s *CodedCharSetId* attribute; see page 267 for details of this attribute.

The following values are defined:

CSQM
Queue manager’s coded character set identifier.

Character data in the application message data is in the queue manager’s character set.

CSEMBD
Embedded coded character set identifiers.

The coded character-set identifier for character data in the message is embedded within the application message data itself. There can be any number of character-set identifiers embedded within the message, applying to different parts of the message.

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 value CSQM to the value of the queue manager's *CodedCharSetId* attribute; as a result, the value CSQM is never returned by the MQGET 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 GMCONV 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 CSQM 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 CSQM.

MDFMT (8-byte character string)
Format name.

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” 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:

FMNONE
No format name.

The nature of the application message data is undefined. This means that the data cannot be converted when the message is retrieved from a queue.

Note: If GMCONV is specified on the MQGET call for a message that has a format name of FMNONE, and the character set or encoding of the message differs from that specified in the

MSGDSC parameter, the message is still returned in the *BUFFER* parameter (assuming no other errors), but the call completes with completion code CCWARN and reason code RC2110.

FMADMN
: 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 GMCONV option is specified on the MQGET call. Refer to the *MQSeries Programmable Command Formats* for more information about programmable command format.

FMCMD1
: 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 GMCONV option is specified on the MQGET call.

FMCMD2
: 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 GMCONV option is specified on the MQGET call.

FMDLH
: 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 *DLFMT* field in the MQDLH structure; see page 19 for details of this structure. Messages of this format can be converted if the GMCONV option is specified on the MQGET call.

  COA and COD reports are not generated for messages which have a *MDFMT* of FMDLH.

FMDH
: 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 *DHFMT* field in the MQDH structure; see page 13 for details of this structure. Messages with format FMDH can be converted if the GMCONV option is specified on the MQGET call.

FMEVNT
: Event message.

  The message is an MQ event message that reports an event that occurred. Messages of this format can be converted if the GMCONV option is specified on the MQGET call. Event messages have the same structure as programmable commands; refer to the *MQSeries*

Programmable Command Formats for more information about this structure.

FMIMS
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 `IIFMT` field in the MQIIH structure. Messages of this format can be converted if the GMCONV option is specified on the MQGET call.

FMIMVS
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 `MDENC` field.

`zz` is a 2-byte field containing flags that are significant to IMS. `zz` is a byte string consisting of two 1-byte bit string 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 the `MDCSI` field.

Messages of this format can be converted if the GMCONV option is specified on the MQGET call.

FMMDE
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 `MEFMT`, `MECSI`, and `MEENC` fields in the MQMDE. See page 104 for details of this structure. Messages of this format can be converted if the GMCONV option is specified on the MQGET call.

FMPCF
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 GMCONV option is specified on the MQGET call. Refer to the *MQSeries Programmable Command Formats* for more information about programmable command format.

FMRMH
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 *RMFMT*, *RMCSI*, and *RMENC* fields in the MQRMH. See page 104 for details of this structure. Messages of this format can be converted if the GMCONV option is specified on the MQGET call.

FMSTR
: 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 GMCONV option is specified on the MQGET call.

FMTM
: Trigger message.

  The message is a trigger message, described by the MQTM structure; see page 151 for details of this structure. Messages of this format can be converted if the GMCONV option is specified on the MQGET call.

FMXQH
: 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 *MDFMT* field in the MQMD structure which is part of the transmission queue header MQXQH. See page 159 for details of this structure.

  COA and COD reports are not generated for messages which have a *MDFMT* of FMXQH.

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 LNFMT. The initial value of this field is FMNONE.

MDPRI (10-digit signed integer)
: 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:

  PRQDEF
  : Default priority for queue.

    The priority for the message is taken from the *DefPriority* attribute for the destination queue, as defined at the local queue manager. The value of *DefPriority* is copied into the *MDPRI* field when the message is put. If *DefPriority* is changed subsequently, messages that have already been put are not affected.

    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 (even if this is a queue-manager alias).

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 (given by the *MaxPriority* queue-manager attribute—see page 270), 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 CCWARN and reason code RC2049. However, the *MDPRI* 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 PRQDEF 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 PRQDEF.

MDPER (10-digit signed integer)
Message persistence.

For the MQPUT and MQPUT1 calls, the value must be one of the following:

PEPER
Message is persistent.

The message survives restarts of the queue manager. Because temporary dynamic queues *do not* survive restarts of the queue manager, persistent messages cannot be put on temporary dynamic queues; persistent messages can however be put on permanent dynamic queues, and predefined queues.

Once a persistent message has been put (or the unit of work committed, if the put request is part of a unit of work), the message is available on auxiliary storage until such time as the message is removed from the queue (or the unit of work committed, if the get request is 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.

PENPER
Message is not persistent.

The message does not survive restarts of the queue manager. This applies even if an intact copy of the message is found on auxiliary storage during the restart procedure.

PEQDEF
Message has default persistence.

The persistence for the message is taken from the *DefPersistence* attribute for the destination queue, as defined at the local queue manager. The value of *DefPersistence* is copied into the *MDPER* field when the message is put. If *DefPersistence* is changed subsequently, messages that have already been put are not affected.

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 (even if this is a queue-manager alias).

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. In other situations, defaulting to the queue definition allows persistence to be changed without changing the application.

For an MQGET call, the value returned is either PEPER or PENPER.

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 PEQDEF.

MDMID (24-byte bit string)
Message identifier.

For the MQGET call, this field specifies the message identifier of the message to be retrieved. Normally the queue manager returns the first message whose *MDMID* and *MDCID* fields match exactly those contained in the message descriptor specified on the MQGET call. However, the special values MINONE and CINONE can be used as global substitutes:

- Specifying MINONE as the message identifier on the MQGET call allows *any* message identifier to match.
- Specifying CINONE as the correlation identifier on the MQGET call allows *any* correlation identifier to match.

Thus it is possible to select messages based on:

- *MDMID* alone
- *MDCID* alone
- Both *MDMID* and *MDCID*
- No criteria

In each case, the message returned is the *first* message that satisfies the selection criteria, or if no criteria, the first message on the queue. But if GMBRWN is specified, the message returned is the *next* message in the sequence satisfying the selection criteria.

Note: The queue is scanned sequentially for a message which satisfies the criteria, so retrieval times will be slower than if no criteria are specified, especially if many messages have to be scanned before a suitable one is found.

This field is ignored if the GMMUC option is specified in the *GMO* parameter on the MQGET call.

On return from an MQGET call, the *MDMID* field is set to the message identifier of the message returned (if any).

For the MQPUT and MQPUT1 calls, if MINONE is specified by the application, the queue manager generates a unique message identifier[1]

[1] A *MDMID* 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

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 required, but the value of the *MDMID* field in MQMD is unchanged on return from the call, even if MINONE was specified. If the application needs to know the message identifiers generated by the queue manager, the application must provide MQPMR records containing the *PRMID* field.

The sending application can also specify a particular value for the message identifier, other than MINONE; this stops the queue manager generating a unique message identifier. This facility can be used by an application that is forwarding a message, 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 issued the get request for the message
- Copy the value to the *MDCID* of any report message that it generates about this message (depending on the *MDREP* options)

When the queue manager or a message channel agent generates a report message, it sets the *MDMID* in the way specified by the *MDREP* field of the original message, either RONMI or ROPMI. Applications that generate report messages should also do this.

The message identifier value associated with a PEPER message persists across restarts of the queue manager.

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 following special value may be used:

MINONE
: No message identifier is specified.

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

This is an input/output field for the MQGET, MQPUT, and MQPUT1 calls. The length of this field is given by LNMID. The initial value of this field is MINONE.

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.

MDCID (24-byte bit string)
Correlation identifier.

For the MQGET call, this field specifies the correlation identifier of the message to be retrieved. See *MDMID* for details on how to specify values for this field.

If the GMMUC option is specified in the *GMO* parameter on the MQGET call, this field is ignored.

On return from an MQGET call, the *MDCID* field is set to the correlation identifier of the message returned (if any).

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 issued the get request for the message.

When the queue manager or a message channel agent generates a report message, it sets the *MDCID* in the way specified by the *MDREP* field of the original message, either ROCMTC or ROPCI. Applications which generate report messages should also do this.

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 following special value may be used:

CINONE
No correlation identifier is specified.

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

For the MQGET call, this is an input/output field. For the MQPUT and MQPUT1 calls, this is an input field if PMNCID is *not* specified, and an output field if PMNCID *is* specified. The length of this field is given by LNCID. The initial value of this field is CINONE.

MDBOC (10-digit signed integer)
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 the GMBRWF or GMBRWN options.

The accuracy of this count is affected by the *HardenGetBackout* local queue attribute; see page 251.

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.

MDRQ (48-byte character string)
Name of reply-to queue.

This is the name of the message queue to which the application that issued the get request for the message should send MTRPLY and MTRPRT messages. The name is the local name of a queue that is defined on the queue manager identified by *MDRM*. 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 *MDMT* field has the value MTRQST, or if any reports are requested by the *MDREP* 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 *MDRM* field is blank, the local queue manager looks up the *MDRQ* name in its own queue definitions. If a local definition of a remote queue exists with this name, the *MDRQ* 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, *MDRQ* 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 *MDRQ* 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 *MDRQ* field should be set to blanks; 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 on page 268).

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 LNQN. The initial value of this field is 48 blank characters.

MDRM (48-byte character string)
Name of reply queue manager.

This is the name of the queue manager to which the reply message or report message should be sent. *MDRQ* is the local name of a queue that is defined on this queue manager.

If the *MDRM* field is blank, the local queue manager looks up the *MDRQ* name in its queue definitions. If a local definition of a remote queue exists with this name, the *MDRM* 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 *MDRM* 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 *MDRM* 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 *MDRM* field should be set to blanks; 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 LNQMN. The initial value of this field is 48 blank characters.

MDUID (12-byte character string)
User identifier.

This is part of the **identity context** of the message; it identifies the user that originated this message. This information can be used in the *ODAU* field of the *OBJDSC* parameter when opening an object, so that the authorization check is performed for the *MDUID* user instead of the application performing the open.

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 uses the *ODAU* from the *OBJDSC* parameter if OOALTU was specified on the corresponding MQOPEN call (or if PMALTU is specified with the MQPUT1 call). Otherwise, it uses a user identifier determined by the environment:

- On OS/400, it uses the name of the signed-on user profile.

For the MQPUT and MQPUT1 calls, this is an input/output field if PMSETI or PMSETA is specified in the *PMO* 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 PMSETI or PMSETA 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 *MDUID* 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 LNUID. The initial value of this field is 12 blank characters.

MDACC (32-byte bit string)
Accounting token.

This is part of the **identity context** of the message; it 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 sets:

- The first byte of the field to the length of the accounting information present in the remainder of the field; this length is in the range zero through 31, and is stored in the first byte as a binary integer.

- The second and subsequent bytes, as indicated by the length field, to the accounting information appropriate to the environment.
  - On OS/400, the accounting information is set to the accounting code for the job.
- All remaining bytes are set to binary zero.

For the MQPUT and MQPUT1 calls, this is an input/output field if PMSETI or PMSETA is specified in the *PMO* parameter. If neither PMSETI nor PMSETA 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 *MDACC* 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:

ACNONE
No accounting token is specified.

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

The length of this field is given by LNACCT. The initial value of this field is ACNONE.

MDAID (32-byte character string)
Application data relating to identity.

This is part of the **identity context** of the message; it is information that is defined by the application suite, and can be used to provide additional information about the message or its originator.

The 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 PMSETI or PMSETA is specified in the *PMO* parameter. If a null character is present, the null and any following characters are converted to blanks by the queue manager. If neither PMSETI nor PMSETA 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 *MDAID* 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 LNAIDD. The initial value of this field is 32 blank characters.

MDPAT (10-digit signed integer)
Type of application that put the message.

This is part of the **origin context** of the message. For more information on message context, see the *MQSeries Application Programming Guide*.

It may have one of the following standard types. User-defined types can also be used but should be restricted to values in the range ATUFST through ATULST.

ATAIX
AIX application (same value as ATUNIX).

ATCICS
CICS transaction.

ATDOS
DOS client application.

ATIMS
IMS application.

ATIMSB
IMS bridge.

ATMVS
MVS or TSO application.

ATOS2
OS/2 or Presentation Manager application.

AT400
OS/400 application.

ATQM
Queue-manager-generated message.

ATUNIX
UNIX application.

ATWIN
Windows client or 16-bit Windows application.

ATWINT
Windows NT or 32-bit Windows application.

ATXCF
XCF.

ATDEF
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.

ATUNK
Unknown application type.

This value can be used to indicate that the application type is unknown, even though other context information is present.

ATUFST
Lowest value for user-defined application type.

ATULST
: Highest value for user-defined application type.

The following special value can also occur:

ATNCON
: 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 PMNOC context option is specified).

When a message is retrieved, *MDPAT* can be tested for this value to decide whether the message has context (it is recommended that *MDPAT* is never set to ATNCON, by an application using PMSETA, if any of the other context fields are nonblank).

When the queue manager generates this information as a result of an application put, the field is set to a value that is determined by the environment. Note that on OS/400, it is set to AT400; the queue manager never uses ATCICS on OS/400.

For the MQPUT and MQPUT1 calls, this is an input/output field if PMSETA is specified in the *PMO* parameter. If PMSETA 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 *MDPAT* that was transmitted with the message. If the message has no context, the field is set to ATNCON.

This is an output field for the MQGET call. The initial value of this field is ATNCON.

MDPAN (28-byte character string)
: Name of application that put the message.

This is part of the **origin context** of the message. The format of the name depends on the *MDPAT*. For more information on message context, see the *MQSeries Application Programming Guide*.

When this field is set by the queue manager, (that is, for all options except PMSETA), it is set to value which is determined by the environment:

- On OS/400, the queue manager uses the fully-qualified job name.

For the MQPUT and MQPUT1 calls, this is an input/output field if PMSETA is specified in the *PMO* 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 PMSETA 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 *MDPAN* 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 LNPAN. The initial value of this field is 28 blank characters.

MDPD (8-byte character string)
: Date when message was put.

This is part of the **origin context** of the message. For more information on message context, 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 *MDPD* and *MDPT* fields, subject to the system clock being set accurately to GMT.

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 PMSETA is specified in the *PMO* 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 PMSETA 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 *MDPD* 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 LNPDAT. The initial value of this field is 8 blank characters.

MDPT (8-byte character string)
Time when message was put.

This is part of the **origin context** of the message. For more information on message context, 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 *MDPT*. This happens when leap seconds are inserted into the global time standard.

Greenwich Mean Time (GMT) is used for the *MDPD* and *MDPT* fields, subject to the system clock being set accurately to 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 PMSETA is specified in the *PMO* 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 PMSETA 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 *MDPT* 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 LNPTIM. The initial value of this field is 8 blank characters.

MDAOD (4-byte character string)
Application data relating to origin.

This is part of the **origin context** of the message; it 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 suitably authorized applications to indicate whether the identity data is trusted. For more information on message context, see the *MQSeries Application Programming Guide*.

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 PMSETA is specified in the *PMO* 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 PMSETA 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 *MDAOD* 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 LNAORD. The initial value of this field is 4 blank characters.

The remaining fields in this structure are not present if *MDVER* is less than MDVER2.

MDGID (24-byte bit string)
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. *MDGID* is also used if segmentation is allowed for the message. In all of these cases, *MDGID* has a non-null value, and one or more of the following flags is set in the *MDMFL* field:

MFMIG
MFLMIG
MFSEG
MFLSEG
MFSEGA

If none of these flags is set, *MDGID* has the special null value GINONE.

This field need not be set by the application on the MQPUT or MQGET call if:

- On the MQPUT call, PMLOGO is specified.
- On the MQGET call, MOGRPI 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 *MDGID* 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 PMLOGO 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 PMLOGO is *not* specified, the application should request the queue manager to generate the group identifier, by setting *MDGID* to GINONE 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 PMLOGO 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 24 on page 125. 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 *PRGID* field.

On input to the MQGET call, the queue manager uses the value detailed in Table 9 on page 40 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:

GINONE
: 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.

The length of this field is given by LNGID. The initial value of this field is GINONE. This field is not present if *MDVER* is less than MDVER2.

MDSEQ (10-digit signed integer)
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, PMLOGO is specified.
- On the MQGET call, MOSEQN 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 *MDSEQ* is set to an appropriate value.

On input to the MQPUT and MQPUT1 calls, the queue manager uses the value detailed in Table 24 on page 125. 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 9 on page 40 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 not present if *MDVER* is less than MDVER2.

MDOFF (10-digit signed integer)
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, PMLOGO is specified.
- On the MQGET call, MOOFFS 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 *MDOFF* is set to an appropriate value.

On input to the MQPUT and MQPUT1 calls, the queue manager uses the value detailed in Table 24 on page 125. 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 *MDOLN* field (provided it is not OLUNDF) 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 9 on page 40 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 not present if *MDVER* is less than MDVER2.

MDMFL (10-digit signed integer)
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 flag: 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 GMCMPM option on the MQGET call, and supplying a buffer that is big enough to accommodate the complete message. (See page 43 for details of the GMCMPM 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:

MFSEGI
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.

MFSEGA
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. MFSEGA can be set without either MFSEG or MFLSEG being set.

The queue manager splits messages into segments as necessary in order to ensure that the segments (plus any header data that may be required) will fit on the queue. The queue-manager does the following:

- User-defined formats are split on boundaries which are multiples of 16 bytes.

- Built-in formats other than FMSTR 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. Hence a second or later segment generated by the queue manager can begin only with one of the following:
  - An MQ header structure
  - The start of the application message data
  - Part-way through the application message data
- FMSTR 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 *MDFMT*, *MDCSI*, and *MDENC* 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 *MDREP* field in the MQMD of segments with *MDOFF* greater than zero are modified as follows:
  - For each report type, if the report option is RO*D, 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 RO*.

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 RC2255.

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 GMCMPM 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 FMSTR and the character set is DBCS or mixed SBCS/DBCS, the sending application should itself create and put the segments, specifying MFSEGI 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 *GMSEG* field in MQGMO.

The initial value of this flag is MFSEGI.

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:

MFMIG
: Message is a member of a group.

MFLMIG
: Message is the last logical message in a group.

  If this flag is set, the queue manager turns on MFMIG 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, MFLMIG is set, but the *MDSEQ* field has the value one.

MFSEG
: Message is a segment of a logical message.

  When MFSEG is specified without MFLSEG, 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 RC2253.

MFLSEG
Message is the last segment of a logical message.

If this flag is set, the queue manager turns on MFSEG 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, MFLSEG is set, but the *MDOFF* field has the value zero.

When MFLSEG 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 PMLOGO 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 PMLOGO is specified:

- If there is no current group or logical message, all of these flags (and combinations of them) are valid.
- Once MFMIG has been specified, it must remain on until MFLMIG is specified. The call fails with reason code RC2241 if this condition is not satisfied.
- Once MFSEG has been specified, it must remain on until MFLSEG is specified. The call fails with reason code RC2242 if this condition is not satisfied.
- Once MFSEG has been specified without MFMIG, MFMIG must remain *off* until after MFLSEG has been specified. The call fails with reason code RC2242 if this condition is not satisfied.

Table 24 on page 125 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 *GMGST* and *GMSST* fields in MQGMO.

Default flags: The following can be specified to indicate that the message has default attributes:

MFNONE
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. MFNONE 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 *MDMFL* field is partitioned into subfields; for details see Appendix C, "Report options" on page 379.

The initial value of this field is MFNONE. This field is not present if *MDVER* is less than MDVER2.

MDOLN (10-digit signed integer)
Length of original message.

This field is of relevance only for report messages; it specifies the length of the message to which the report relates. If the report message is reporting on a segment, *MDOLN* is the length of the segment, and *not* the length of the logical message of which the segment forms part, nor the length of the data in the report message.

MDOLN should be set by the program which generates the report, or which segments the original message, but if that program does not set the field, *MDOLN* has the following special value:

OLUNDF
Original length of message not defined.

This is an input field on the MQPUT and MQPUT1 calls, but the value provided by the application is used only in particular circumstances:

- 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.
- If the message being put is a report message reporting on a segment, 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 RC2252.
- In all other cases, the queue manager ignores the field and uses the value OLUNDF instead.

This is an output field on the MQGET call.

The initial value of this field is OLUNDF. This field is not present if *MDVER* is less than MDVER2.

Table 15. Initial values of fields in MQMD

Field name	Name of constant	Value of constant
MDSID	MDSIDV	'MDbb' (See note 1)
MDVER	MDVER1	1
MDREP	RONONE	0
MDMT	MTDGRM	8
MDEXP	EIULIM	-1
MDFB	FBNONE	0
MDENC	ENNAT	See note 2
MDCSI	CSQM	0
MDFMT	FMNONE	'bbbbbbbb'
MDPRI	PRQDEF	-1
MDPER	PEQDEF	2
MDMID	MINONE	Nulls
MDCID	CINONE	Nulls
MDBOC	None	0
MDRQ	None	Blanks
MDRM	None	Blanks
MDUID	None	Blanks
MDACC	ACNONE	Nulls
MDAID	None	Blanks
MDPAT	ATNCON	0
MDPAN	None	Blanks
MDPD	None	Blanks
MDPT	None	Blanks
MDAOD	None	Blanks
MDGID	GINONE	Nulls
MDSEQ	None	1
MDOFF	None	0
MDMFL	MFNONE	0
MDOLN	OLUNDF	-1

Notes:

1. The symbol ‘b’ represents a single blank character.
2. The value of this constant is environment-specific.

RPG declaration (ILE)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQMD Structure
D*
D* Structure identifier
D  MDSID                  1      4
D* Structure version number
D  MDVER                  5      8I 0
D* Report options
D  MDREP                  9     12I 0
D* Message type
D  MDMT                  13     16I 0
D* Expiry time
D  MDEXP                 17     20I 0
D* Feedback or reason code
D  MDFB                  21     24I 0
D* Data encoding
D  MDENC                 25     28I 0
D* Coded character set identifier
D  MDCSI                 29     32I 0
D* Format name
D  MDFMT                 33     40
D* Message priority
D  MDPRI                 41     44I 0
D* Message persistence
D  MDPER                 45     48I 0
D* Message identifier
D  MDMID                 49     72
D* Correlation identifier
D  MDCID                 73     96
D* Backout counter
D  MDBOC                 97    100I 0
D* Name of reply-to queue
D  MDRQ                 101    148
D* Name of reply queue manager
D  MDRM                 149    196
D* User identifier
D  MDUID                197    208
D* Accounting token
D  MDACC                209    240
D* Application data relating to identity
D  MDAID                241    272
D* Type of application that put the message
D  MDPAT                273    276I 0
D* Name of application that put the message
D  MDPAN                277    304
D* Date when message was put
D  MDPD                 305    312
D* Time when message was put
D  MDPT                 313    320
D* Application data relating to origin
D  MDAOD                321    324
D* Group identifier
D  MDGID                325    348
D* Sequence number of logical message within group
D  MDSEQ                349    352I 0
D* Offset of data in physical message from start of logical message
```

```
D  MDOFF                353    356I 0
D* Message flags
D  MDMFL                357    360I 0
D* Length of original message
D  MDOLN                361    364I 0
```

RPG declaration (OPM)

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7..
I* MQMD Structure
I*
I* Structure identifier
I                                        1   4 MDSID
I* Structure version number
I                                     B   5   80MDVER
I* Report options
I                                     B   9  120MDREP
I* Message type
I                                     B  13  160MDMT
I* Expiry time
I                                     B  17  200MDEXP
I* Feedback or reason code
I                                     B  21  240MDFB
I* Data encoding
I                                     B  25  280MDENC
I* Coded character set identifier
I                                     B  29  320MDCSI
I* Format name
I                                       33  40 MDFMT
I* Message priority
I                                     B  41  440MDPRI
I* Message persistence
I                                     B  45  480MDPER
I* Message identifier
I                                       49  72 MDMID
I* Correlation identifier
I                                       73  96 MDCID
I* Backout counter
I                                     B  97 1000MDBOC
I* Name of reply-to queue
I                                      101 148 MDRQ
I* Name of reply queue manager
I                                      149 196 MDRM
I* User identifier
I                                      197 208 MDUID
I* Accounting token
I                                      209 240 MDACC
I* Application data relating to identity
I                                      241 272 MDAID
I* Type of application that put the message
I                                     B 273 2760MDPAT
I* Name of application that put the message
I                                      277 304 MDPAN
I* Date when message was put
I                                      305 312 MDPD
I* Time when message was put
I                                      313 320 MDPT
I* Application data relating to origin
```

```
I                                     321 324 MDAOD
I* Group identifier
I                                     325 348 MDGID
I* Sequence number of logical message within group
I                                   B 349 3520MDSEQ
I* Offset of data in physical message from start of logical message
I                                   B 353 3560MDOFF
I* Message flags
I                                   B 357 3600MDMFL
I* Length of original message
I                                   B 361 3640MDOLN
```

MQMDE – Message descriptor extension

The following table guides you to the appropriate page for each field.

Table 16. Fields in MQMDE

This field...	Describes...	See page ...
MESID	Structure identifier	106
MEVER	Structure version number	106
MELEN	Structure length	107
MEENC	Data encoding	107
MECSI	Coded character-set identifier	107
MEFMT	Format name	107
MEFLG	General flags	107
MEGID	Group identifier	107
MESEQ	Sequence number of logical message within group	108
MEOFF	Offset of data in physical message from start of logical message	108
MEMFL	Message flags	108
MEOLN	Length of original message	108

The MQMDE structure describes the data that sometimes occurs preceding the application message data. 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 contains those MQMD fields that exist in the version-2 MQMD, but not in the version-1 MQMD. It 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 *MDFMT* field in MQMD to FMMDE 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 18 on page 108.

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 17 on page 105.

There is one special case. If the application uses a version-2 MQMD to put a message that is a segment (that is, the MFSEG or MFLSEG flag is set), and the format name in the MQMD is FMDLH, 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.

Table 17. 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, and same as MQMD	MQMDE is honored
2	Not default	Valid, but different from MQMD	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

The data in the MQMDE structure must be in the queue manager's character set and encoding. The former is given by the *CodedCharSetId* queue-manager attribute (see page 267), while in most cases the latter is given by the value of ENNAT. If this condition is not satisfied, 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 ENNAT that is different from the queue-manager's encoding. Although numeric fields in the MQMD structure on the MQPUT, MQPUT1, and MQGET calls must be in the Micro Focus COBOL encoding, numeric fields in the MQMDE structure must be in the queue-manager's encoding. This latter is given by ENNAT for the C programming language, and has the value 546.

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 *MDFMT* field in MQMD will have the value FMMDE 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 will have been 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 will usually be in the queue manager's character set and encoding. The one exception is if the MQMDE was treated as data on the MQPUT or MQPUT1 call (see Table 17 on page 105 for the circumstances that can cause this).

Note: On OS/2 and Windows NT, applications compiled with Micro Focus COBOL use a value of ENNAT 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

MESID (4-byte character string)
Structure identifier.

The value must be:

MESIDV
Identifier for message descriptor extension structure.

The initial value of this field is MESIDV.

MEVER (10-digit signed integer)
Structure version number.

The value must be:

MEVER2
: Version-2 message descriptor extension structure.

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

MEVERC
: Current version of message descriptor extension structure.

The initial value of this field is MEVER2.

MELEN (10-digit signed integer)
Length of MQMDE structure.

The following value is defined:

MELEN2
: Length of version-2 message descriptor extension structure.

The initial value of this field is MELEN2.

MEENC (10-digit signed integer)
Encoding of the data following the MQMDE.

The queue manager does not check the value of this field. See the description of the *MDENC* field in the MQMD structure on page 77 for more information about data encodings.

The initial value of this field is ENNAT.

MECSI (10-digit signed integer)
Character-set identifier of the data following the MQMDE.

The queue manager does not check the value of this field.

The initial value of this field is 0.

MEFMT (8-byte character string)
Format name of the data following the MQMDE.

The queue manager does not check the value of this field. See the description of the *MDFMT* field in the MQMD structure on page 78 for more information about format names.

The initial value of this field is FMNONE.

MEFLG (10-digit signed integer)
General flags.

The following flag can be specified:

MEFNON
: No flags.

The initial value of this field is MEFNON.

MEGID (24-byte bit string)
Group identifier.

See the *MDGID* field in the MQMD structure on page 92. The initial value of this field is GINONE.

MESEQ (10-digit signed integer)
Sequence number of logical message within group.

See the *MDSEQ* field in the MQMD structure on page 94. The initial value of this field is 1.

MEOFF (10-digit signed integer)
Offset of data in physical message from start of logical message.

See the *MDOFF* field in the MQMD structure on page 94. The initial value of this field is 0.

MEMFL (10-digit signed integer)
Message flags.

See the *MDMFL* field in the MQMD structure on page 95. The initial value of this field is MFNONE.

MEOLN (10-digit signed integer)
Length of original message.

See the *MDOLN* field in the MQMD structure on page 99. The initial value of this field is OLUNDF.

Table 18. Initial values of fields in MQMDE

Field name	Name of constant	Value of constant
MESID	MESIDV	'MDEb' (See note 1)
MEVER	MEVER2	2
MELEN	MELEN2	72
MEENC	ENNAT	See note 2
MECSI	None	0
MEFMT	FMNONE	'bbbbbbbb'
MEFLG	MEFNON	0
MEGID	GINONE	Nulls
MESEQ	None	1
MEOFF	None	0
MEMFL	MFNONE	0
MEOLN	OLUNDF	-1

Notes:

1. The symbol ‘b’ represents a single blank character.
2. The value of this constant is environment-specific.

RPG declaration (ILE)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQMDE Structure
D*
D* Structure identifier
D  MESID                 1      4
D* Structure version number
D  MEVER                 5      8I 0
D* Length of MQMDE structure
```

```
D  MELEN                 9     12I 0
D* Encoding of the data following the MQMDE
D  MEENC                13     16I 0
D* Character-set identifier of the data following the MQMDE
D  MECSI                17     20I 0
D* Format name of the data following the MQMDE
D  MEFMT                21     28
D* General flags
D  MEFLG                29     32I 0
D* Group identifier
D  MEGID                33     56
D* Sequence number of logical message within group
D  MESEQ                57     60I 0
D* Offset of data in physical message from start of logical message
D  MEOFF                61     64I 0
D* Message flags
D  MEMFL                65     68I 0
D* Length of original message
D  MEOLN                69     72I 0
```

RPG declaration (OPM)

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7..
I* MQMDE Structure
I*
I* Structure identifier
I                                        1   4 MESID
I* Structure version number
I                                    B   5   80MEVER
I* Length of MQMDE structure
I                                    B   9  120MELEN
I* Encoding of the data following the MQMDE
I                                    B  13  160MEENC
I* Character-set identifier of the data following the MQMDE
I                                    B  17  200MECSI
I* Format name of the data following the MQMDE
I                                       21  28 MEFMT
I* General flags
I                                    B  29  320MEFLG
I* Group identifier
I                                       33  56 MEGID
I* Sequence number of logical message within group
I                                    B  57  600MESEQ
I* Offset of data in physical message from start of logical message
I                                    B  61  640MEOFF
I* Message flags
I                                    B  65  680MEMFL
I* Length of original message
I                                    B  69  720MEOLN
```

MQOD – Object descriptor

The following table guides you to the appropriate page for each field.

Table 19. Fields in MQOD

This field...	Describes...	See page ...
ODSID	Structure identifier	111
ODVER	Structure version number	111
ODOT	Object type	111
ODON	Object name	111
ODMN	Object queue manager name	112
ODDN	Dynamic queue name	112
ODAU	Alternate user identifier	113
Note: The remaining fields are supported only in the version-2 structure.		
ODREC	Number of object records present	113
ODKDC	Number of local queues opened successfully	113
ODUDC	Number of remote queues opened successfully	113
ODIDC	Number of queues that failed to open	114
ODORO	Offset of first object record	114
ODRRO	Offset of first response record	114
ODORP	Address of first object record	115
ODRRP	Address of first response record	115

The MQOD structure is used to specify an object by name. The following types of object are valid:

- Queue or distribution list
- Process definition
- Queue manager

The current version of MQOD is ODVER2. Fields that exist only in the version-2 structure are identified as such in the descriptions that follow. The declaration of MQOD provided in the COPY file contains the new fields, but the initial value provided for the *ODVER* field is ODVER1; this ensures compatibility with existing applications. To use the new fields, the application must set the version number to ODVER2. Applications which are intended to be portable between several environments should use a version-2 MQOD only if all of those environments support version 2.

To open a distribution list, a version-2 MQOD must be used.

This structure is an input/output parameter for the MQOPEN and MQPUT1 calls.

Fields

ODSID (4-byte character string)
Structure identifier.

The value must be:

ODSIDV
Identifier for object descriptor structure.

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

ODVER (10-digit signed integer)
Structure version number.

The value must be one of the following:

ODVER1
Version-1 object descriptor structure.

ODVER2
Version-2 object descriptor structure.

Fields that exist only in the version-2 structure are identified as such in the descriptions that follow.

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

ODVERC
Current version of object descriptor structure.

This is always an input field. The initial value of this field is ODVER1.

ODOT (10-digit signed integer)
Object type.

Type of object being named in *ODON*. Possible values are:

OTQ
Queue.
OTPRO
Process definition.
OTQM
Queue manager.

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

ODON (48-byte character string)
Object name.

The local name of the object as defined on the queue manager identified by *ODMN*.

The name must not contain leading or embedded blanks, but may contain trailing blanks; the first null character and characters following it are treated as blanks. For more information about names, see the *MQSeries Application Programming Guide*.

If *ODOT* is OTQM, special rules apply; in this case the name must be entirely blank up to the first null character or the end of the field.

If *ODON* 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

ODON field the name of the queue created. A model queue can be specified only for the MQOPEN call.

If a distribution list is being opened (that is, *ODREC* is present and greater than zero), *ODON* must be blank or the null string. If this condition is not satisfied, the call fails with reason code RC2152.

This is an input/output field for the MQOPEN call when *ODON* is the name of a model queue, and an input-only field in all other cases. The length of this field is given by LNQN. The initial value of this field is 48 blank characters.

ODMN (48-byte character string)
Object queue manager name.

This is the name of the queue manager on which the *ODON* object is defined.

If the name is specified, it must not contain leading or embedded blanks, but may contain trailing blanks; the first null character and characters following it are treated as blanks.

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.

If *ODOT* is OTQM, the name of the local queue manager must either be specified explicitly, or specified as blank.

If *ODON* 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 *ODMN* 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 for the MQOPEN call.

If a distribution list is being opened (that is, *ODREC* is greater than zero), *ODMN* must be blank or the null string. If this condition is not satisfied, the call fails with reason code RC2153.

This is an input/output field for the MQOPEN call when *ODON* is the name of a model queue, and an input-only field in all other cases. The length of this field is given by LNQMN. The initial value of this field is 48 blank characters.

ODDN (48-byte character string)
Dynamic queue name.

This is an input field that is ignored unless *ODON* specifies the name of a model queue. If it does, this field specifies the name of the dynamic queue to be created.

The name must not contain leading or embedded blanks, but may contain trailing blanks; the first null character and characters following it are treated as blanks. A completely blank name (or one in which only blanks appear before the first null character) is not valid if *ODON* specifies 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.

The length of this field is given by LNQN. The initial value of this field is

ODAU (12-byte character string)
Alternate user identifier.

If OOALTU is specified for the MQOPEN call, or PMALTU 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 OOALTU or PMALTU 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 OOALTU nor PMALTU is specified, this field is ignored.

This is an input field. The length of this field is given by LNUID. The initial value of this field is 12 blank characters.

ODREC (10-digit signed integer)
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 *ODREC* being the number of destination queues in the list. It is valid for a distribution list to contain only one destination.

The value of *ODREC* must not be less than zero, and if it is greater than zero *ODOT* must be OTQ; the call fails with reason code RC2154 if these conditions are not satisfied.

This is an input field. The initial value of this field is 0. This field is not present if *ODVER* is less than ODVER2.

ODKDC (10-digit signed integer)
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 not present if *ODVER* is less than ODVER2.

ODUDC (10-digit signed integer)
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 not present if *ODVER* is less than ODVER2.

ODIDC (10-digit signed integer)
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 *CMPCOD* parameter on the MQOPEN or MQPUT1 call is CCOK or CCWARN; it is *not* set if the *CMPCOD* parameter is CCFAIL.

This is an output field. The initial value of this field is 0. This field is not present if *ODVER* is less than ODVER2.

ODORO (10-digit signed integer)
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. *ODORO* is used only when a distribution list is being opened. The field is ignored if *ODREC* 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 *ODORO*

  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 *ODORO* 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.

- By using the pointer field *ODORP*

  In this case, the application can declare the array of MQOR structures separately from the MQOD structure, and set *ODORP* to the address of the array.

Whichever technique is chosen, one of *ODORO* and *ODORP* must be used; the call fails with reason code RC2155 if both are zero, or both are nonzero.

This is an input field. The initial value of this field is 0. This field is not present if *ODVER* is less than ODVER2.

ODRRO (10-digit signed integer)
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. *ODRRO* is used only when a distribution list is being opened. The field is ignored if *ODREC* 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 (*RRCC* field in MQRR), and the reason for each failure (*RRREA*

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 RC2136 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 *ODREC* of them.

The response records can be provided in the same way as the object records, either by specifying an offset in *ODRRO*, or by specifying an address in *ODRRP*; see the description of *ODORO* above for details of how to do this. However, no more than one of *ODRRO* and *ODRRP* can be used; the call fails with reason code RC2156 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 CCOK or CCWARN.

This is an input field. The initial value of this field is 0. This field is not present if *ODVER* is less than ODVER2.

ODORP (pointer)

Address of first object record.

This is the address of the first MQOR object record. *ODORP* is used only when a distribution list is being opened. The field is ignored if *ODREC* is zero.

Either *ODORP* or *ODORO* can be used to specify the object records, but not both; see the description of the *ODORO* field above for details. If *ODORP* 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. This field is not present if *ODVER* is less than ODVER2.

ODRRP (pointer)

Address of first response record.

This is the address of the first MQRR response record. *ODRRP* is used only when a distribution list is being opened. The field is ignored if *ODREC* is zero.

Either *ODRRP* or *ODRRO* can be used to specify the response records, but not both; see the description of the *ODRRO* field above for details. If *ODRRP* 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. This field is not present if *ODVER* is less than ODVER2.

Table 20. Initial values of fields in MQOD

Field name	Name of constant	Value of constant
ODSID	ODSIDV	'ODbb' (See note 1)
ODVER	ODVER1	1
ODOT	OTQ	1
ODON	None	Blanks
ODMN	None	Blanks
ODDN	None	
ODAU	None	Blanks
ODREC	None	0
ODKDC	None	0
ODUDC	None	0
ODIDC	None	0
ODORO	None	0
ODRRO	None	0
ODORP	None	Null pointer or null bytes
ODRRP	None	Null pointer or null bytes

Notes:

1. The symbol ‘b’ represents a single blank character.

RPG declaration (ILE)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQOD Structure
D*
D* Structure identifier
D  ODSID                  1      4
D* Structure version number
D  ODVER                  5      8I 0
D* Object type
D  ODOT                   9     12I 0
D* Object name
D  ODON                  13     60
D* Object queue manager name
D  ODMN                  61    108
D* Dynamic queue name
D  ODDN                 109    156
D* Alternate user identifier
D  ODAU                 157    168
D* Number of object records present
D  ODREC                169    172I 0
D* Number of local queues opened successfully
D  ODKDC                173    176I 0
D* Number of remote queues opened successfully
D  ODUDC                177    180I 0
D* Number of queues that failed to open
D  ODIDC                181    184I 0
```

```
D* Offset of first object record from start of MQOD
D  ODORO                185    188I 0
D* Offset of first response record from start of MQOD
D  ODRRO                189    192I 0
D* Address of first object record
D  ODORP                193    208*
D* Address of first response record
D  ODRRP                209    224*
```

RPG declaration (OPM)

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7..
I* MQOD Structure
I*
I* Structure identifier
I                                        1   4 ODSID
I* Structure version number
I                                    B   5   80ODVER
I* Object type
I                                    B   9  120ODOT
I* Object name
I                                       13  60 ODON
I* Object queue manager name
I                                       61 108 ODMN
I* Dynamic queue name
I                                      109 156 ODDN
I* Alternate user identifier
I                                      157 168 ODAU
I* Number of object records present
I                                    B 169 1720ODREC
I* Number of local queues opened successfully
I                                    B 173 1760ODKDC
I* Number of remote queues opened successfully
I                                    B 177 1800ODUDC
I* Number of queues that failed to open
I                                    B 181 1840ODIDC
I* Offset of first object record from start of MQOD
I                                    B 185 1880ODORO
I* Offset of first response record from start of MQOD
I                                    B 189 1920ODRRO
I* Address of first object record
I                                      193 208 ODORP
I* Address of first response record
I                                      209 224 ODRRP
```

MQOR – Object records

The following table guides you to the appropriate page for each field.

Table 21. Fields in MQOR

This field...	Describes...	See page ...
ORON	Object name	118
ORMN	Object queue manager name	118

The MQOR structure is used to specify the queue name and queue-manager name of a single destination queue. 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.

The character data in the MQOR structure must be in the queue-manager's character set. MQOR is an input structure for the MQOPEN and MQPUT1 calls.

Fields

ORON (48-byte character string)

Object name.

This is the same as the *ODON* 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 48 blank characters.

ORMN (48-byte character string)

Object queue manager name.

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

This is always an input field. The initial value of this field is 48 blank characters.

Table 22. Initial values of fields in MQOR

Field name	Name of constant	Value of constant
ORON	None	Blanks
ORMN	None	Blanks

RPG declaration (ILE)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQOR Structure
D*
D* Object name
D  ORON                    1     48
D* Object queue manager name
D  ORMN                   49     96
```

RPG declaration (OPM)

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7..
I* MQOR Structure
I*
I* Object name
I                                       1  48 ORON
I* Object queue manager name
I                                      49  96 ORMN
```

MQPMO – Put message options

The following table guides you to the appropriate page for each field.

Table 23. Fields in MQPMO

This field...	Describes...	See page ...
PMSID	Structure identifier	121
PMVER	Structure version number	121
PMOPT	Options that control the action of MQPUT and MQPUT1	121
PMCT	Object handle of input queue	130
PMKDC	Number of messages sent successfully to local queues	130
PMUDC	Number of messages sent successfully to remote queues	130
PMIDC	Number of messages that could not be sent	130
PMRQN	Resolved name of destination queue	130
PMRMN	Resolved name of destination queue manager	131
Note: The remaining fields are supported only in the version-2 structure.		
PMREC	Number of put message records or response records present	131
PMPRF	Flags for put message record fields	131
PMPRO	Offset of first put message record	132
PMRRO	Offset of first response record	133
PMPRP	Address of first put message record	134
PMRRP	Address of first response record	134

The current version of MQPMO is PMVER2. Fields that exist only in the version-2 structure are identified as such in the descriptions that follow. The declaration of MQPMO provided in the COPY file contains the new fields, but the initial value provided for the *PMVER* field is PMVER1; this ensures compatibility with existing applications. To use the new fields, the application must set the version number to PMVER2. Applications which are intended to be portable between several environments should use a version-2 MQPMO only if all of those environments support version 2.

The MQPMO structure is an input/output parameter for the MQPUT and MQPUT1 calls.

Fields

PMSID (4-byte character string)
Structure identifier.

The value must be:

PMSIDV
Identifier for put-message options structure.

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

PMVER (10-digit signed integer)
Structure version number.

The value must be one of the following:

PMVER1
Version-1 put-message options structure.

PMVER2
Version-2 put-message options structure.

Fields that exist only in the version-2 structure are identified as such in the descriptions that follow.

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

PMVERC
Current version of put-message options structure.

This is always an input field. The initial value of this field is PMVER1.

PMOPT (10-digit signed integer)
Options that control the action of MQPUT or 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). Combinations that are not valid are noted; any other combinations are valid.

The following options are described; The page on which each description can be found is shown in the following table:

Option	See page
PMSYP	122
PMNSYP	122
PMNMID	122
PMNCID	122
PMLOGO	123
PMNOC	127
PMDEFC	128
PMPASI	128
PMPASA	128
PMSETI	128
PMSETA	129
PMALTU	129
PMFIQ	129
PMNONE	129

PMSYP
: 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 PMNSYP is specified, the put request is not within a unit of work.

  PMSYP must *not* be specified with PMNSYP.

PMNSYP
: 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 PMSYP is specified, the put request is not within a unit of work.

  PMNSYP must *not* be specified with PMSYP.

PMNMID
: Generate a new message identifier.

  This option causes the queue manager to replace the contents of the *MDMID* 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 *PRMID* field in the MQPMR structure for details.

  Using this option relieves the application of the need to reset the *MDMID* field to MINONE prior to each MQPUT or MQPUT1 call.

PMNCID
: Generate a new correlation identifier.

  This option causes the queue manager to replace the contents of the *MDCID* 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 *PRCID* field in the MQPMR structure for details.

  PMNCID is useful in situations where the application requires a unique correlation identifier.

Group and segment option: The option described below relates to messages in groups and segments of logical messages. The following definitions may be of help in understanding this 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 (*MDMID* 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 (*MDGID* field in MQMD), and the same message sequence number (*MDSEQ* field in MQMD). The segments are distinguished by differing values for the segment offset (*MDOFF* 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 (GINONE), 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.

PMLOGO

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 PMLOGO 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 *MDGID*, *MDSEQ*, and *MDOFF* fields in MQMD, as the queue manager sets these to the appropriate values. The application need set only the the *MDMFL* 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 MF* flags in *MDMFL* 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 RC2241 or RC2242, 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 MFLMIG and/or MFLSEG as appropriate, before reissuing the MQPUT call to put the message that is not in the group or not a segment.

Table 24 on page 125 shows the combinations of options and flags that are valid, and the values of the *MDGID*, *MDSEQ*, and *MDOFF* 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 abbreviated column headings denote the following options, flags, and group and logical-message status:

- **LOG ORD** means the PMLOGO option.
- **MIG** means the MFMIG and/or MFLMIG flag.
- **SEG** means the MFSEG and/or MFLSEG flag.
- **SEG OK** means the MFSEGA flag.
- **Cur grp** means that a current message group exists prior to the call.
- **Cur log msg** means that a current logical message exists prior to the call.

In the table:

- “(√)” indicates that the row applies whether or not there is a √ in that column.
- “Previous” denotes the value used for that field in the previous message for the queue handle.

Table 24. 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**	*MDGID*	*MDSEQ*	*MDOFF*
√						GINONE	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
				(√)	(√)	GINONE	1	0
			√	(√)	(√)	New group id if GINONE, else value in field	1	0
		√	(√)	(√)	(√)	New group id if GINONE, else value in field	1	Value in field
	√		(√)	(√)	(√)	New group id if GINONE, else value in field	Value in field	0
	√	√	(√)	(√)	(√)	New group id if GINONE, else value in field	Value in field	Value in field

Notes:

- PMLOGO is not valid on the MQPUT1 call.
- For the *MDMID* field, the queue manager generates a new message identifier if PMNMID or MINONE is specified, and uses the value in the field otherwise.
- For the *MDCID* field, the queue manager generates a new correlation identifier if PMNCID is specified, and uses the value in the field otherwise.

When PMLOGO 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 *MDPER* 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 RC2185.

The PMLOGO 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 RC2245.

When PMLOGO is specified, the MQMD supplied on the MQPUT call must not be less than MDVER2. If this condition is not satisfied, the call fails with reason code RC2257.

If PMLOGO 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 *MDGID*, *MDSEQ*, *MDOFF*, and *MDMFL* 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 *MDGID*, *MDSEQ*, *MDOFF*, *MDMFL*, and *MDPER* fields to the appropriate values, and then issue the MQPUT call with PMSYP or PMNSYP set as desired, but *without* specifying PMLOGO. If this call is successful, the queue manager retains the group and segment information, and subsequent MQPUT calls using that queue handle can specify PMLOGO 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 PMLOGO with MQPUT calls that do not, but the following points should be noted:

- Each successful MQPUT call that does *not* specify PMLOGO 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 PMLOGO 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 CCWARN completion code. Table 25 on page 127 shows the various cases that can arise. In these cases, if the completion code is not CCOK, the reason code is one of the following (as appropriate):

  RC2241
  RC2242
  RC2185
  RC2245

Note: The queue manager does not check the group and segment information for the MQPUT1 call.

Table 25. Outcome when MQPUT or MQCLOSE call not consistent with group and segment information

Current call	Previous call	
	MQPUT with PMLOGO	MQPUT without PMLOGO
MQPUT with PMLOGO	CCFAIL	CCFAIL
MQPUT without PMLOGO	CCWARN	CCOK
MQCLOSE with an unterminated group or logical message	CCWARN	CCOK

Applications that simply want to put messages and segments in logical order are recommended to specify PMLOGO, 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 PMLOGO option, and this can be achieved by not specifying that option. If this is done, the application must ensure that the *MDGID*, *MDSEQ*, *MDOFF*, and *MDMFL* 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 PMLOGO. There are two reasons for this:

- If the messages are retrieved and put in order, specifying PMLOGO 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 PMLOGO, nor the corresponding GMLOGO 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 PMLOGO when putting the report message.

PMLOGO can be specified with any of the other PM* options.

PMNOC

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

PMDEFC
: 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
MDUID	Determined from the environment if possible; set to blanks otherwise.
MDACC	Determined from the environment if possible; set to ACNONE otherwise.
MDAID	Set to blanks.
MDPAT	Determined from the environment.
MDPAN	Determined from the environment if possible; set to blanks otherwise.
MDPD	Set to date when message is put.
MDPT	Set to time when message is put.
MDAOD	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.

PMPASI
: 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 *PMCT* field. Origin context information is generated by the queue manager in the same way that it is for PMDEFC (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 OOPASI 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 OOPASI option.

PMPASA
: 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 *PMCT* 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 OOPASA 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 OOPASA option.

PMSETI
: 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 PMDEFC (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 OOSETI 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 OOSETI option.

PMSETA
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 OOSETA option. For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the OOSETA option.

Only one of the PM* context options can be specified. If none of these options is specified, PMDEFC is assumed.

PMALTU
Validate with specified user identifier.

This indicates that the *ODAU* field in the *OBJDSC* 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 *ODAU* 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.

PMFIQ
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.

The call returns completion code CCFAIL with reason code RC2161.

PMNONE
No options specified.

This value can be used to indicate that no other options have been specified; all options assume their default values. PMNONE 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 *PMOPT* field is PMNONE.

PMTO (10-digit signed integer)
Reserved.

This is a reserved field; its value is not significant. The initial value of this field is −1.

PMCT (10-digit signed integer)
Object handle of input queue.

If PMPASI or PMPASA 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 PMPASI nor PMPASA is specified, this field is ignored.

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

PMKDC (10-digit signed integer)
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 *PMVER* is less than PMVER2.

PMUDC (10-digit signed integer)
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 *PMVER* is less than PMVER2.

PMIDC (10-digit signed integer)
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 *CMPCOD* parameter on the MQPUT or MQPUT1 call is CCOK or CCWARN; it is *not* set if the *CMPCOD* parameter is CCFAIL.

This is an output field. The initial value of this field is 0. This field is not set if *PMVER* is less than PMVER2.

PMRQN (48-byte character string)
Resolved name of destination queue.

This is an output field that is set by the queue manager to the name of the queue (after alias resolution) on which the message will be placed. This can be either the name of a local queue, or the name of a remote queue. If the destination queue opened was a model queue, the name of the dynamic local queue that was created is returned. In all cases, the name

returned is the name of a queue that is defined on the queue manager identified by *PMRMN*.

If the MQPUT or MQPUT1 call is used to put the message to a distribution list, the value returned in this field is undefined.

This is an output field. The length of this field is given by LNQN. The initial value of this field is 48 blank characters.

PMRMN (48-byte character string)
Resolved name of destination queue manager.

This is the name of the queue manager (after alias resolution) that owns the queue specified by *PMRQN*.

If the MQPUT or MQPUT1 call is used to put the message to a distribution list, the value returned in this field is undefined.

This is an output field. The length of this field is given by LNQMN. The initial value of this field is 48 blank characters.

The remaining fields in this structure are not present if *PMVER* is less than PMVER2.

PMREC (10-digit signed integer)
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 *PMREC* records of each type.

The value of *PMREC* 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 *PMPRO* below).

If *PMREC* 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 RC2154.

This is an input field. The initial value of this field is 0. This field is not present if *PMVER* is less than PMVER2.

PMPRF (10-digit signed integer)
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. *PMPRF* is used only when the message is being put to a distribution list. The field is ignored if *PMREC* is zero, or both *PMPRO* and *PMPRP* 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:

PFMID
: Message-identifier field is present.

PFCID
: Correlation-identifier field is present.

PFGID
: Group-identifier field is present.

PFFB
: Feedback field is present.

PFACC
: Accounting-token field is present.

  If this flag is specified, either PMSETI or PMSETA must be specified in the *PMOPT* field; if this condition is not satisfied, the call fails with reason code RC2158.

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

PFNONE
: No put-message record fields are present.

  If this value is specified, either *PMREC* must be zero, or both *PMPRO* and *PMPRP* must be zero.

  PFNONE 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 *PMPRF* contains flags which are not valid, or put message records are provided but *PMPRF* has the value PFNONE, the call fails with reason code RC2158.

This is an input field. The initial value of this field is PFNONE. This field is not present if *PMVER* is less than PMVER2.

PMPRO (10-digit signed integer)
: 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. *PMPRO* is used only when the message is being put to a distribution list. The field is ignored if *PMREC* 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 *PMREC* 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 *PMPRO*, or by specifying an address in *PMPRP*; see the description of *ODORO* on page 114 for details of how to do this.

No more than one of *PMPRO* and *PMPRP* can be used; the call fails with reason code RC2159 if both are nonzero.

This is an input field. The initial value of this field is 0. This field is not present if *PMVER* is less than PMVER2.

PMRRO (10-digit signed integer)

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. *PMRRO* is used only when the message is being put to a distribution list. The field is ignored if *PMREC* 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 (*RRCC* field in MQRR), and the reason for each failure (*RRREA* 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 RC2136 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 *PMREC* of them.

The response records can be provided in a similar way to the object records in MQOD, either by specifying an offset in *PMRRO*, or by specifying an address in *PMRRP*; see the description of *ODORO* on page 114 for details of how to do this. However, no more than one of *PMRRO* and *PMRRP* can be used; the call fails with reason code RC2156 if both are nonzero.

For the MQPUT1 call, this field must be the null pointer or 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 not present if *PMVER* is less than PMVER2.

PMPRP (pointer)
: Address of first put message record.

  This is the address of the first MQPMR put message record. *PMPRP* is used only when the message is being put to a distribution list. The field is ignored if *PMREC* is zero.

  Either *PMPRP* or *PMPRO* can be used to specify the put message records, but not both; see the description of the *PMPRO* field above for details. If *PMPRP* 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. This field is not present if *PMVER* is less than PMVER2.

PMRRP (pointer)
: Address of first response record.

  This is the address of the first MQRR response record. *PMRRP* is used only when the message is being put to a distribution list. The field is ignored if *PMREC* is zero.

  Either *PMRRP* or *PMRRO* can be used to specify the response records, but not both; see the description of the *PMRRO* field above for details. If *PMRRP* 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. This field is not present if *PMVER* is less than PMVER2.

Table 26. Initial values of fields in MQPMO

Field name	Name of constant	Value of constant
PMSID	PMSIDV	'PMOb' (See note 1)
PMVER	PMVER1	1
PMOPT	PMNONE	0
PMTO	None	-1
PMCT	None	0
PMKDC	None	0
PMUDC	None	0
PMIDC	None	0
PMRQN	None	Blanks
PMRMN	None	Blanks
PMREC	None	0
PMPRF	None	0
PMPRO	None	0
PMRRO	None	0
PMPRP	None	Null pointer or null bytes
PMRRP	None	Null pointer or null bytes

Notes:

1. The symbol 'b' represents a single blank character.

RPG declaration (ILE)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQPMO Structure
D*
D* Structure identifier
D  PMSID                   1      4
D* Structure version number
D  PMVER                   5      8I 0
D* Options that control the action of MQPUT or MQPUT1
D  PMOPT                   9     12I 0
D* Reserved
D  PMTO                   13     16I 0
D* Object handle of input queue
D  PMCT                   17     20I 0
D* Number of messages sent successfully to local queues
D  PMKDC                  21     24I 0
D* Number of messages sent successfully to remote queues
D  PMUDC                  25     28I 0
D* Number of messages that could not be sent
D  PMIDC                  29     32I 0
D* Resolved name of destination queue
D  PMRQN                  33     80
D* Resolved name of destination queue manager
D  PMRMN                  81    128
```

```
D* Number of put message records or response records present
D  PMREC                 129    132I 0
D* Flags indicating which MQPMR fields are present
D  PMPRF                 133    136I 0
D* Offset of first put message record from start of MQPMO
D  PMPRO                 137    140I 0
D* Offset of first response record from start of MQPMO
D  PMRRO                 141    144I 0
D* Address of first put message record
D  PMPRP                 145    160*
D* Address of first response record
D  PMRRP                 161    176*
```

RPG declaration (OPM)

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7..
I* MQPMO Structure
I*
I* Structure identifier
I                                        1   4 PMSID
I* Structure version number
I                                    B   5   80PMVER
I* Options that control the action of MQPUT or MQPUT1
I                                    B   9  120PMOPT
I* Reserved
I                                    B  13  160PMTO
I* Object handle of input queue
I                                    B  17  200PMCT
I* Number of messages sent successfully to local queues
I                                    B  21  240PMKDC
I* Number of messages sent successfully to remote queues
I                                    B  25  280PMUDC
I* Number of messages that could not be sent
I                                    B  29  320PMIDC
I* Resolved name of destination queue
I                                       33  80 PMRQN
I* Resolved name of destination queue manager
I                                       81 128 PMRMN
I* Number of put message records or response records present
I                                    B 129 1320PMREC
I* Flags indicating which MQPMR fields are present
I                                    B 133 1360PMPRF
I* Offset of first put message record from start of MQPMO
I                                    B 137 1400PMPRO
I* Offset of first response record from start of MQPMO
I                                    B 141 1440PMRRO
I* Address of first put message record
I                                      145 160 PMPRP
I* Address of first response record
I                                      161 176 PMRRP
```

MQPMR – Put message records

The following table guides you to the appropriate page for each field.

Table 27. Fields in MQPMR

This field...	Describes...	See page ...
PRMID	Message identifier	137
PRCID	Correlation identifier	138
PRGID	Group identifier	138
PRFB	Feedback code	138
PRACC	Accounting token	138

The MQPMR structure is used to specify various message properties for a single destination. 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 *PMPRF* field in MQPMO. Fields that are present ***must occur in the order shown below***. Fields that are absent occupy no space in the record.

Because MQPMR does not have a fixed layout, no declaration is provided for it in a COPY file. The application programmer should create a declaration containing the fields that are required by the application, and set the flags in *PMPRF* to indicate the fields that are present.

MQPMR is an input/output structure for the MQPUT and MQPUT1 calls.

Fields

PRMID (24-byte bit string)

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 *MDMID* 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 *PRMID* field. If that value is MINONE, a new message identifier is generated for *each* of those destinations (that is, no two of those destinations have the same message identifier).

If PMNMID 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 PMNCID is processed (see below).

This is an input/output field.

PRCID (24-byte bit string)
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 *MDCID* 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 *PRCID* field.

If PMNCID 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 PMNMID is processed (see above).

This is an input/output field.

PRGID (24-byte bit string)
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 *MDGID* 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 *PRGID* field. The value is processed as documented in Table 24 on page 125, 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 RC2258.

This is an input/output field.

PRFB (10-digit signed integer)
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 *MDFB* 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.

PRACC (32-byte bit string)
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 *MDACC* 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.

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.

RPG declaration (ILE)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQPMR Structure
D*
D* Message identifier
D  PRMID                  1     24
D* Correlation identifier
D  PRCID                 25     48
D* Group identifier
D  PRGID                 49     72
D* Feedback or reason code
D  PRFB                  73     76I 0
D* Accounting token
D  PRACC                 77    108
```

RPG declaration (OPM)

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7..
I* MQPMR Structure
I*
I* Message identifier
I                                         1  24 PRMID
I* Correlation identifier
I                                        25  48 PRCID
I* Group identifier
I                                        49  72 PRGID
I* Feedback or reason code
I                                    B   73  760PRFB
I* Accounting token
I                                        77 108 PRACC
```

MQRMH –Reference message header

The following table guides you to the appropriate page for each field.

Table 28. Fields in MQRMH

This field...	Describes...	See page ...
RMSID	Structure identifier	141
RMVER	Structure version number	141
RMLEN	The total length of MQRMH	141
RMENC	Data encoding	142
RMCSI	Coded character set identifier	142
RMFMT	Format name	142
RMFLG	Control flags	142
RMOT	Type of object	143
RMOII	Object instance identifier	143
RMSEL	Length of source environment data	143
RMSEO	Offset of source environment data	143
RMSNL	Length of source object name	143
RMSNO	Offset of source object name	144
RMDEL	Length of destination environment data	144
RMDEO	Offset of destination environment data	144
RMDNL	Length of destination object name	144
RMDNO	Offset of destination object name	144
RMDL	Length of bulk data	145
RMDO	Low offset of bulk data	145
RMDO2	High offset of bulk data	146

The MQRMH structure defines the format of a reference message header. An application can put a message in this format, 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 (optionally with the bulk data) 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.

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 *RMLEN* 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 *RMFMT* field in MQRMH has a value other than FMNONE.
- 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 GMCONV option.

The format name of an MQRMH structure is FMRMH. The fields in the MQRMH structure, and the strings addressed by the offset fields, are in the character set and encoding given by the *MDCSI* and *MDENC* fields in the header structure that precedes the MQRMH, or by those fields in the MQMD structure if the MQRMH is at the start of the application message data.

Fields

RMSID (4-byte character string)
Structure identifier.

The value must be:

RMSIDV
Identifier for reference message header structure.

The initial value of this field is RMSIDV.

RMVER (10-digit signed integer)
Structure version number.

The value must be:

RMVER1
Version-1 reference message header structure.

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

RMVERC
Current version of reference message header structure.

The initial value of this field is RMVER1.

RMLEN (10-digit signed integer)
Total length of MQRMH, including strings at end of fixed fields, but not the bulk data.

The initial value of this field is zero.

RMENC (10-digit signed integer)
Data encoding.

This identifies the representation used for numeric values in the bulk data; this applies to binary integer data, packed-decimal integer data, and floating-point data.

The initial value of this field is ENNAT.

RMCSI (10-digit signed integer)
Coded character set identifier.

This specifies the coded character set identifier of character data in the bulk data.

Note that character data in the MQ data structures must be in the character set used by the queue manager. This is defined by the queue manager's *CodedCharSetId* attribute; see page 267 for details of this attribute.

The initial value of this field is 0.

RMFMT (8-byte character string)
Format name.

This is a name that the sender of the message may use to indicate to the receiver the nature of the bulk data. 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. Do not use a null character to terminate the name before the end of the field, as the queue manager does not change the null and subsequent characters to blanks in the MQRMH structure. Do not specify a name with leading or embedded blanks.

The initial value of this field is FMNONE.

RMFLG (10-digit signed integer)
Reference message flags.

The following flags are defined:

RMLAST
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.

RMNLST
Reference message does not contain or represent last part of object.

RMNLST 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 RMNLST.

RMOT (8-byte character string)
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 *RMFMT* field described above.

The initial value of this field is 8 blanks.

RMOII (24-byte bit string)
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:

OIINON
No object instance identifier specified.

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

The length of this field is given by LNOIID. The initial value of this field is OIINON.

RMSEL (10-digit signed integer)
Length of source environment data.

If this field is zero, there is no source environment data, and *RMSEO* is ignored.

The initial value of this field is 0.

RMSEO (10-digit signed integer)
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 *RMSEL*; if this length is zero, there is no source environment data, and *RMSEO* is ignored. If present, the source environment data must reside completely within *RMLEN* 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 *RMSNO*, *RMDEO*, and *RMDNO* fields.

The initial value of this field is 0.

RMSNL (10-digit signed integer)
Length of source object name.

If this field is zero, there is no source object name, and *RMSNO* is ignored.

The initial value of this field is 0.

RMSNO (10-digit signed integer)
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 *RMSNL*; if this length is zero, there is no source object name, and *RMSNO* is ignored. If present, the source object name must reside completely within *RMLEN* 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 *RMSEO*, *RMDEO*, and *RMDNO* fields.

The initial value of this field is 0.

RMDEL (10-digit signed integer)
Length of destination environment data.

If this field is zero, there is no destination environment data, and *RMDEO* is ignored.

RMDEO (10-digit signed integer)
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 *RMDEL*; if this length is zero, there is no destination environment data, and *RMDEO* is ignored. If present, the destination environment data must reside completely within *RMLEN* 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 *RMSEO*, *RMSNO*, and *RMDNO* fields.

The initial value of this field is 0.

RMDNL (10-digit signed integer)
Length of destination object name.

If this field is zero, there is no destination object name, and *RMDNO* is ignored.

RMDNO (10-digit signed integer)
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 *RMDNL*; if this length is zero, there is no destination object name, and *RMDNO* is ignored. If present, the destination object name must reside completely within *RMLEN* 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 *RMSEO*, *RMSNO*, and *RMDEO* fields.

The initial value of this field is 0.

RMDL (10-digit signed integer)
Length of bulk data.

The *RMDL* 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 *RMLEN* bytes from the start of the MQRMH structure. The length of the entire message minus *RMLEN* gives the length of the bulk data present.

If data is present in the message, *RMDL* specifies the amount of that data that is relevant. The normal case is for *RMDL* 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 *RMDL*, 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.

RMDO (10-digit signed integer)
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 *RMLEN*.

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:

- *RMDO* 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.
- *RMDO2* 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.

RMDO2 (10-digit signed integer)
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 *RMDO* for details.

The initial value of this field is 0.

Table 29. Initial values of fields in MQRMH

Field name	Name of constant	Value of constant
RMSID	RMSIDV	'RMHb' (See note 1)
RMVER	RMVER1	1
RMLEN	None	0
RMENC	ENNAT	See note 2
RMCSI	None	0
RMFMT	FMNONE	'bbbbbbbb'
RMFLG	RMNLST	0
RMOT	None	'bbbbbbbb'
RMOII	OIINON	Nulls
RMSEL	None	0
RMSEO	None	0
RMSNL	None	0
RMSNO	None	0
RMDEL	None	0
RMDEO	None	0
RMDNL	None	0
RMDNO	None	0
RMDL	None	0
RMDO	None	0
RMDO2	None	0

Notes:

1. The symbol 'b' represents a single blank character.
2. The value of this constant is environment-specific.

RPG declaration (ILE)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQRMH Structure
D*
D* Structure identifier
D  RMSID                  1      4
D* Structure version number
D  RMVER                  5      8I 0
D* Total length of MQRMH, including strings at end of fixed fields,
D* but not the bulk data
D  RMLEN                  9     12I 0
```

```
D* Data encoding
D  RMENC                 13     16I 0
D* Coded character set identifier
D  RMCSI                 17     20I 0
D* Format name
D  RMFMT                 21     28
D* Reference message flags
D  RMFLG                 29     32I 0
D* Object type
D  RMOT                  33     40
D* Object instance identifier
D  RMOII                 41     64
D* Length of source environment data
D  RMSEL                 65     68I 0
D* Offset of source environment data
D  RMSEO                 69     72I 0
D* Length of source object name
D  RMSNL                 73     76I 0
D* Offset of source object name
D  RMSNO                 77     80I 0
D* Length of destination environment data
D  RMDEL                 81     84I 0
D* Offset of destination environment data
D  RMDEO                 85     88I 0
D* Length of destination object name
D  RMDNL                 89     92I 0
D* Offset of destination object name
D  RMDNO                 93     96I 0
D* Length of bulk data
D  RMDL                  97    100I 0
D* Low offset of bulk data
D  RMDO                 101    104I 0
D* High offset of bulk data
D  RMDO2                105    108I 0
```

RPG declaration (OPM)

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7..
I* MQRMH Structure
I*
I* Structure identifier
I                                        1   4 RMSID
I* Structure version number
I                                     B   5   80RMVER
I* Total length of MQRMH, including strings at end of fixed fields,
I* but not the bulk data
I                                     B   9  120RMLEN
I* Data encoding
I                                     B  13  160RMENC
I* Coded character set identifier
I                                     B  17  200RMCSI
I* Format name
I                                       21  28 RMFMT
I* Reference message flags
I                                     B  29  320RMFLG
I* Object type
I                                       33  40 RMOT
I* Object instance identifier
```

```
I                                     41  64 RMOII
I* Length of source environment data
I                                  B  65  680RMSEL
I* Offset of source environment data
I                                  B  69  720RMSEO
I* Length of source object name
I                                  B  73  760RMSNL
I* Offset of source object name
I                                  B  77  800RMSNO
I* Length of destination environment data
I                                  B  81  840RMDEL
I* Offset of destination environment data
I                                  B  85  880RMDEO
I* Length of destination object name
I                                  B  89  920RMDNL
I* Offset of destination object name
I                                  B  93  960RMDNO
I* Length of bulk data
I                                  B  97 1000RMDL
I* Low offset of bulk data
I                                  B 101 1040RMDO
I* High offset of bulk data
I                                  B 105 1080RMDO2
```

MQRR – Response records

The following table guides you to the appropriate page for each field.

Table 30. Fields in MQRR

This field...	Describes...	See page ...
RRCC	Completion code	149
RRREA	Reason code	149

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. 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 RC2136 from the call indicates that the response records (if provided by the application) have been set by the queue manager.

MQRR is an output structure for the MQOPEN, MQPUT, and MQPUT1 calls.

Fields

RRCC (10-digit signed integer)

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 CCOK.

RRREA (10-digit signed integer)

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.

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

Table 31. Initial values of fields in MQRR

Field name	Name of constant	Value of constant
RRCC	CCOK	0
RRREA	RCNONE	0

RPG declaration (ILE)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQRR Structure
D*
D* Completion code for queue
D  RRCC                   1      4I 0
D* Reason code for queue
D  RRREA                  5      8I 0
```

RPG declaration (OPM)

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7..
I* MQRR Structure
I*
I* Completion code for queue
I                                     B   1   40RRCC
I* Reason code for queue
I                                     B   5   80RRREA
```

MQTM – Trigger message

The following table guides you to the appropriate page for each field.

Table 32. Fields in MQTM

This field...	Describes...	See page ...
TMSID	Structure identifier	152
TMVER	Structure version number	152
TMQN	Name of triggered queue	152
TMPN	Name of process object	153
TMTD	Trigger data	153
TMAT	Application type	153
TMAI	Application identifier	153
TMED	Environment data	154
TMUD	User data	154

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.

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 *TMQN*, *TMTD*, and *TMUD*. The trigger monitor application can pass the MQTM structure directly to the started application, or pass an MQTMC structure, depending on what is most convenient for the started application. For information about MQTMC, see page 156.

For information about triggers, see the *MQSeries Application Programming Guide*.

The fields in the message descriptor of the trigger message are set as follows:

Field in MQMD	**Value used**
MDSID	MDSIDV
MDVER	MDVER1
MDREP	RONONE
MDMT	MTDGRM
MDEXP	EIULIM
MDFB	FBNONE
MDENC	ENNAT
MDCSI	Queue manager's *CodedCharSetId* attribute
MDFMT	FMTM
MDPRI	Initiation queue's *DefPriority* attribute
MDPER	PENPER
MDMID	A unique value
MDCID	CINONE
MDBOC	0
MDRQ	Blanks
MDRM	Name of queue manager

MDUID	Blanks
MDACC	ACNONE
MDAID	Blanks
MDPAT	ATQM, or as appropriate for the message channel agent
MDPAN	First 28 bytes of the queue-manager name
MDPD	Date when trigger message is sent
MDPT	Time when trigger message is sent
MDAOD	Blanks

An application that generates a trigger message is recommended to set similar values, except for the following:

- The *MDPRI* field can be set to PRQDEF (the queue manager will change this to the default priority for the initiation queue when the message is put).
- The *MDRM* 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

TMSID (4-byte character string)
Structure identifier.

The value must be:

TMSIDV
Identifier for trigger message structure.

The initial value of this field is TMSIDV.

TMVER (10-digit signed integer)
Structure version number.

The value must be:

TMVER1
Version number for trigger message structure.

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

TMVERC
Current version of trigger message structure.

The initial value of this field is TMVER1.

TMQN (48-byte character string)
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 *QName* on page 246 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 LNQN. The initial value of this field is 48 blank characters.

TMPN (48-byte character string)
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 *TMQN* field. See *ProcessName* on page 254 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 LNPRON. The initial value of this field is 48 blank characters.

TMTD (64-byte character string)
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 *TMQN* field. See *TriggerData* on page 258 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 LNTRGD. The initial value of this field is 64 blank characters.

TMAT (10-digit signed integer)
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 *TMPN* field. See *ApplType* on page 263 for details of this attribute. The content of this data is of no significance to the queue manager.

TMAT can have one of the following standard values. User-defined types can also be used, but should be restricted to values in the range ATUFST through ATULST:

ATCICS
CICS transaction.

AT400
OS/400 application.

ATUFST
Lowest value for user-defined application type.

ATULST
Highest value for user-defined application type.

The initial value of this field is 0.

TMAI (256-byte character string)
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 *TMPN* field. See *ApplId* on

page 262 for details of this attribute. The content of this data is of no significance to the queue manager.

The interpretation to be placed on the information is determined by the trigger-monitor application.

The length of this field is given by LNPROA. The initial value of this field is 256 blank characters.

TMED (128-byte character string)
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 *TMPN* field. See *EnvData* on page 263 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 LNPROE. The initial value of this field is 128 blank characters.

TMUD (128-byte character string)
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 *TMPN* field. See *UserData* on page 264 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 LNPROU. The initial value of this field is 128 blank characters.

Table 33. Initial values of fields in MQTM

Field name	Name of constant	Value of constant
TMSID	TMSIDV	'TMbb' (See note 1)
TMVER	TMVER1	1
TMQN	None	Blanks
TMPN	None	Blanks
TMTD	None	Blanks
TMAT	None	0
TMAI	None	Blanks
TMED	None	Blanks
TMUD	None	Blanks

Notes:

1. The symbol ‘b’ represents a single blank character.

RPG declaration (ILE)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQTM Structure
D*
D* Structure identifier
D  TMSID                   1      4
D* Structure version number
D  TMVER                   5      8I 0
D* Name of triggered queue
D  TMQN                    9     56
D* Name of process object
D  TMPN                   57    104
D* Trigger data
D  TMTD                  105    168
D* Application type
D  TMAT                  169    172I 0
D* Application identifier
D  TMAI                  173    428
D* Environment data
D  TMED                  429    556
D* User data
D  TMUD                  557    684
```

RPG declaration (OPM)

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7..
I* MQTM Structure
I*
I* Structure identifier
I                                          1   4 TMSID
I* Structure version number
I                                      B   5   80TMVER
I* Name of triggered queue
I                                          9  56 TMQN
I* Name of process object
I                                         57 104 TMPN
I* Trigger data
I                                        105 168 TMTD
I* Application type
I                                      B 169 1720TMAT
I* Application identifier
I                                        173 428 TMAI
I* Environment data
I                                        429 556 TMED
I* User data
I                                        557 684 TMUD
```

MQTMC – Trigger message (character format)

The following table guides you to the appropriate page for each field.

Table 34. Fields in MQTMC

This field...	Describes...	See page ...
TCSID	Structure identifier	156
TCVER	Structure version number	156
TCQN	Name of triggered queue	156
TCPN	Name of process object	157
TCTD	Trigger data	157
TCAT	Application type	157
TCAI	Application identifier	157
TCED	Environment data	157
TCUD	User data	157

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 *TCQN*, *TCTD*, and *TCUD*. The trigger monitor application should pass a MQTMC structure directly to the started application.

The MQTMC structure is very similar to the format of the trigger message (MQTM structure). The difference is that the noncharacter fields in MQTM are changed in MQTMC to character fields of the same length.

See the description of "MQTM – Trigger message" on page 151 for details of the fields that are the same in this structure.

Fields

TCSID (4-byte character string)
Structure identifier.

The value must be:

TCSIDV
Identifier for trigger message (character format) structure.

TCVER (4-byte character string)
Structure version number.

The value must be:

TCVER1
Version 1 trigger message (character format) structure.

TCQN (48-byte character string)
Name of triggered queue.

See the *TMQN* field in the MQTM structure.

TCPN (48-byte character string)
Name of process object.

See the *TMPN* field in the MQTM structure.

TCTD (64-byte character string)
Trigger data.

See the *TMTD* field in the MQTM structure.

TCAT (4-byte character string)
Application type.

This field always contains blanks, whatever the value in the *TMAT* field in the MQTM structure of the original trigger message.

TCAI (256-byte character string)
Application identifier.

See the *TMAI* field in the MQTM structure.

TCED (128-byte character string)
Environment data.

See the *TMED* field in the MQTM structure.

TCUD (128-byte character string)
User data.

See the *TMUD* field in the MQTM structure.

Table 35. Initial values of fields in MQTMC

Field name	Name of constant	Value of constant
TCSID	TCSIDV	'TMCb' (See note 1)
TCVER	TCVER1	'bbb1'
TCQN	None	Blanks
TCPN	None	Blanks
TCTD	None	Blanks
TCAT	None	'bbbb'
TCAI	None	Blanks
TCED	None	Blanks
TCUD	None	Blanks

Notes:

1. The symbol ‘b’ represents a single blank character.

RPG declaration (ILE)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQTMC Structure
D*
D* Structure identifier
D  TCSID                  1      4
D* Structure version number
D  TCVER                  5      8
D* Name of triggered queue
D  TCQN                   9     56
D* Name of process object
D  TCPN                  57    104
D* Trigger data
D  TCTD                 105    168
D* Application type
D  TCAT                 169    172
D* Application identifier
D  TCAI                 173    428
D* Environment data
D  TCED                 429    556
D* User data
D  TCUD                 557    684
```

RPG declaration (OPM)

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7..
I* MQTMC Structure
I*
I* Structure identifier
I                                         1   4 TCSID
I* Structure version number
I                                         5   8 TCVER
I* Name of triggered queue
I                                         9  56 TCQN
I* Name of process object
I                                        57 104 TCPN
I* Trigger data
I                                       105 168 TCTD
I* Application type
I                                       169 172 TCAT
I* Application identifier
I                                       173 428 TCAI
I* Environment data
I                                       429 556 TCED
I* User data
I                                       557 684 TCUD
```

MQXQH –Transmission queue header

The following table guides you to the appropriate page for each field.

Table 36. Fields in MQXQH

This field...	Describes...	See page ...
XQSID	Structure identifier	162
XQVER	Structure version number	162
XQRQ	Name of destination queue	162
XQRQM	Name of destination queue manager	162
XQMD	Original message descriptor	162

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 USTRAN.

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 a modified version of the message descriptor provided by the application in the *MSGDSC* parameter of the MQPUT or MQPUT1 call (see below for details).

  The message put by the application may be a message in a group, or a segment of a logical message, or may have segmentation allowed, but these properties are *not* propagated into the separate message descriptor – the version-2 fields in the separate message descriptor always have their default values.

  The separate message descriptor is the one that is returned to the application in the *MSGDSC* 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 close copy of the message descriptor that was provided by the application in the *MSGDSC* parameter of the MQPUT or MQPUT1 call (see below for details).

  The embedded message descriptor is always a version-1 MQMD. If the message put by the application has nondefault values for one or more of the version-2 fields in the MQMD, an MQMDE structure follows the MQXQH, and is in turn followed by the application message data (if any). The MQMDE is either:

  – Generated by the queue manager (if the application uses a version-2 MQMD to put the message), or

  – Already present at the start of the application message data (if the application uses a version-1 MQMD to put the message).

  The embedded message descriptor is the one that is returned to the application in the *MSGDSC* parameter of the MQGET call when the message is removed from the final destination queue.

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

Character data in the MQXQH structure is in the character set of the local queue manager (defined by the *CodedCharSetId* queue manager attribute), and integer data is in the native machine encoding. These values are stored in the separate message descriptor, and may be different from the values of the *MDCSI* and *MDENC* fields in the embedded message descriptor, because the latter fields relate to the application message data and not the MQXQH structure itself.

The fields in the embedded message descriptor have the same values as those in the *MSGDSC* parameter of the MQPUT or MQPUT1 call, with the exception of the following:

- The *MDVER* field always has the value MDVER1.
- If the *MDPRI* field has the value PRQDEF, it is replaced by the value of the queue's *DefPriority* attribute.
- If the *MDPER* field has the value PEQDEF, it is replaced by the value of the queue's *DefPersistence* attribute.
- If the *MDMID* field has the value MINONE, or the PMNMID option was specified, or the message is a distribution-list message, *MDMID* 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 *MDMID* field in each of the new embedded message descriptors is the same as that in the original distribution-list message.
- If the PMNCID option was specified, *MDCID* is replaced by a new correlation identifier generated by the queue manager.
- The context fields are set as indicated by the PM* context option(s) specified in the *PMO* parameter; the context fields are the fields *MDUID* through *MDAOD* in the list below.
- 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.

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
MDSID	MDSIDV
MDVER	MDVER2
MDREP	Copied from the embedded message descriptor, but with the bits identified by ROAUXM 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.)
MDMT	Copied from the embedded message descriptor.

Field in separate MQMD	Value used
MDEXP	Copied from the embedded message descriptor.
MDFB	Copied from the embedded message descriptor.
MDENC	ENNAT
MDCSI	Queue manager's *CodedCharSetId* attribute.
MDFMT	FMXQH
MDPRI	Copied from the embedded message descriptor.
MDPER	Copied from the embedded message descriptor.
MDMID	A new value is generated by the queue manager. This message identifier is different from the *MDMID* that the queue manager may have generated for the embedded message descriptor (see above).
MDCID	The *MDMID* from the embedded message descriptor.
MDBOC	0
MDRQ	Copied from the embedded message descriptor.
MDRM	Copied from the embedded message descriptor.
MDUID	Copied from the embedded message descriptor.
MDACC	Copied from the embedded message descriptor.
MDAID	Copied from the embedded message descriptor.
MDPAT	ATQM
MDPAN	First 28 bytes of the queue-manager name.
MDPD	Date when message was put on transmission queue.
MDPT	Time when message was put on transmission queue.
MDAOD	Blanks
MDGID	GINONE
MDSEQ	1
MDOFF	0
MDMFL	MFNONE
MDOLN	OLUNDF

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 *MDFMT* field in the *MSGDSC* parameter of the MQPUT or MQPUT1 call must have the value FMXQH.

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 FMXQH being returned in the *MDFMT* field in the *MSGDSC* parameter of the MQGET call. The values returned in the *MDCSI* and *MDENC* fields in the *MSGDSC* 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 *MDCSI* and *MDENC* fields in the embedded message descriptor.

Fields

XQSID (4-byte character string)
Structure identifier.

The value must be:

XQSIDV
Identifier for transmission-queue header structure.

The initial value of this field is XQSIDV.

XQVER (10-digit signed integer)
Structure version number.

The value must be:

XQVER1
Version number for transmission-queue header structure.

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

XQVERC
Current version of transmission-queue header structure.

The initial value of this field is XQVER1.

XQRQ (48-byte character string)
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 *XQRQM* to be a local definition of another remote queue).

If the message is a distribution-list message (that is, the *MDFMT* field in the embedded message descriptor is FMDH), *XQRQ* is blank.

The length of this field is given by LNQN. The initial value of this field is 48 blank characters.

XQRQM (48-byte character string)
Name of destination queue manager.

This is the name of the queue manager that owns the queue that is the apparent eventual destination for the message.

If the message is a distribution-list message, *XQRQM* is blank.

The length of this field is given by LNQMN. The initial value of this field is 48 blank characters.

XQMD (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 *MSGDSC* 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.

Table 37. Initial values of fields in MQXQH

Field name	Name of constant	Value of constant
XQSID	XQSIDV	'XQHb' (See note 1)
XQVER	XQVER1	1
XQRQ	None	Blanks
XQRQM	None	Blanks
XQMD	Same names and values as for MQMD; see Table 15 on page 100	

Notes:

1. The symbol ‘b’ represents a single blank character.

RPG declaration (ILE)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQXQH Structure
D*
D* Structure identifier
D  XQSID                  1      4
D* Structure version number
D  XQVER                  5      8I 0
D* Name of destination queue
D  XQRQ                   9     56
D* Name of destination queue manager
D  XQRQM                 57    104
D* Original message descriptor
D*
D*      Structure identifier
D  XQ1SID               105    108
D*      Structure version number
D  XQ1VER               109    112I 0
D*      Report options
D  XQ1REP               113    116I 0
D*      Message type
D  XQ1MT                117    120I 0
D*      Expiry time
D  XQ1EXP               121    124I 0
D*      Feedback or reason code
D  XQ1FB                125    128I 0
D*      Data encoding
D  XQ1ENC               129    132I 0
D*      Coded character set identifier
D  XQ1CSI               133    136I 0
D*      Format name
D  XQ1FMT               137    144
D*      Message priority
D  XQ1PRI               145    148I 0
D*      Message persistence
D  XQ1PER               149    152I 0
D*      Message identifier
D  XQ1MID               153    176
```

```
D*     Correlation identifier
D  XQ1CID              177    200
D*     Backout counter
D  XQ1BOC              201    204I 0
D*     Name of reply-to queue
D  XQ1RQ               205    252
D*     Name of reply queue manager
D  XQ1RM               253    300
D*     User identifier
D  XQ1UID              301    312
D*     Accounting token
D  XQ1ACC              313    344
D*     Application data relating to identity
D  XQ1AID              345    376
D*     Type of application that put the message
D  XQ1PAT              377    380I 0
D*     Name of application that put the message
D  XQ1PAN              381    408
D*     Date when message was put
D  XQ1PD               409    416
D*     Time when message was put
D  XQ1PT               417    424
D*     Application data relating to origin
D  XQ1AOD              425    428
```

RPG declaration (OPM)

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7..
I* MQXQH Structure
I*
I* Structure identifier
I                                           1   4 XQSID
I* Structure version number
I                                       B   5   80XQVER
I* Name of destination queue
I                                           9  56 XQRQ
I* Name of destination queue manager
I                                          57 104 XQRQM
I* Original message descriptor
I*
I*     Structure identifier
I                                         105 108 XQ1SID
I*     Structure version number
I                                       B 109 1120XQ1VER
I*     Report options
I                                       B 113 1160XQ1REP
I*     Message type
I                                       B 117 1200XQ1MT
I*     Expiry time
I                                       B 121 1240XQ1EXP
I*     Feedback or reason code
I                                       B 125 1280XQ1FB
I*     Data encoding
I                                       B 129 1320XQ1ENC
I*     Coded character set identifier
I                                       B 133 1360XQ1CSI
I*     Format name
I                                         137 144 XQ1FMT
```

```
I*     Message priority
I                                B 145 1480XQ1PRI
I*     Message persistence
I                                B 149 1520XQ1PER
I*     Message identifier
I                                  153 176 XQ1MID
I*     Correlation identifier
I                                  177 200 XQ1CID
I*     Backout counter
I                                B 201 2040XQ1BOC
I*     Name of reply-to queue
I                                  205 252 XQ1RQ
I*     Name of reply queue manager
I                                  253 300 XQ1RM
I*     User identifier
I                                  301 312 XQ1UID
I*     Accounting token
I                                  313 344 XQ1ACC
I*     Application data relating to identity
I                                  345 376 XQ1AID
I*     Type of application that put the message
I                                B 377 3800XQ1PAT
I*     Name of application that put the message
I                                  381 408 XQ1PAN
I*     Date when message was put
I                                  409 416 XQ1PD
I*     Time when message was put
I                                  417 424 XQ1PT
I*     Application data relating to origin
I                                  425 428 XQ1AOD
```

Chapter 3. Call descriptions

This chapter describes the MQI calls:

- MQCLOSE – Close object
- MQCONN – Connect to queue manager
- 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

Note: The calls associated with data conversion, MQXCNVC and MQDATACONVEXIT, are in Appendix D, "Data-conversion" on page 385.

Conventions used in the call descriptions

For each call, this chapter gives a description of the parameters and usage of the call. This is followed by typical invocations of the call, and typical declarations of its parameters, in the RPG programming language.

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 its direction; for example:

CMPCOD (9-digit decimal integer) — output

There is more information about the structure data types in Chapter 1, "Data type descriptions – elementary" on page 3.

The direction of the parameter can be:

Input You (the programmer) must provide this parameter.

Output The call returns this parameter.

Input/output You must provide this parameter, but it is modified by the call.

There is also a brief description of the purpose of the parameter, together with a list of any values that the parameter can take.

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 Chapter 5, "Return codes" on page 275.

Usage notes
Additional information about the call, describing how to use it and any restrictions on its use.

© Copyright IBM Corp. 1994, 1998

RPG invocation
Typical invocation of the call, and declaration of its parameters, in RPG.

Other notational conventions are:

Constants Names of constants are shown in uppercase; for example, OOOUT.

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.

MQCLOSE – Close object

The MQCLOSE call relinquishes access to an object, and is the inverse of the MQOPEN call.

```
MQCLOSE (HCONN, HOBJ, OPTS, CMPCOD, REASON)
```

Parameters

HCONN (10-digit signed integer) – input
Connection handle.

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN call.

On OS/400, the MQCONN call can be omitted, and the following value specified for *HCONN*:

HCDEFH
Default connection handle.

HOBJ (10-digit signed integer) – 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:

HOUNUH
Unusable object handle.

OPTS (10-digit signed integer) – input
Options that control the action of MQCLOSE.

The *OPTS* 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 QDPERM (see *DefinitionType* on page 249). The close options are summarized in Table 38 on page 171.

One (and only one) of the following must be specified:

CONONE
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 MVS/ESA, if the queue is a dynamic queue that has been logically deleted (see Usage note 3 on page 173), and this is the last handle for it, the queue is physically deleted.

CODEL
: 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 RC2045, and the object is not deleted.

On MVS/ESA, if the queue is a dynamic queue that has been logically deleted (see Usage note 3 on page 173), and this is the last handle for it, the queue is physically deleted.

COPURG
: 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 RC2045, and the object is not deleted.

Table 38. 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	CONONE	CODEL	COPURG
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

CMPCOD (10-digit signed integer) – output
Completion code.

It is one of the following:

CCOK
Successful completion.
CCFAIL
Call failed.

REASON (10-digit signed integer) – output
Reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

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

If *CMPCOD* is CCWARN:

RC2241
(2241, X'8C1') Message group not complete.
RC2242
(2242, X'8C2') Logical message not complete.

If *CMPCOD* is CCFAIL:

RC2219
(2219, X'8AB') MQI call reentered before previous call complete.
RC2009
(2009, X'7D9') Connection to queue manager lost.
RC2018
(2018, X'7E2') Connection handle not valid.
RC2019
(2019, X'7E3') Object handle not valid.
RC2035
(2035, X'7F3') Not authorized for access.
RC2101
(2101, X'835') Object damaged.

RC2045
: (2045, X'7FD') Option not valid for object type.

RC2046
: (2046, X'7FE') Options not valid or not consistent.

RC2058
: (2058, X'80A') Queue manager name not valid or not known.

RC2059
: (2059, X'80B') Queue manager not available for connection.

RC2162
: (2162, X'872') Queue manager shutting down.

RC2055
: (2055, X'807') Queue contains one or more messages or uncommitted put or get requests.

RC2102
: (2102, X'836') Insufficient system resources available.

RC2063
: (2063, X'80F') Security error occurred.

RC2071
: (2071, X'817') Insufficient storage available.

RC2195
: (2195, X'893') Unexpected error occurred.

See Chapter 5, “Return codes” on page 275 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 CONONE option.
2. The following points apply only 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 OOBRW option, the browse cursor is destroyed. If the queue is subsequently reopened with the OOBRW option, a new browse cursor is created (see the OOBRW option on page 206).
   - If a message is currently locked for this handle at the time of the MQCLOSE call, the lock is released (see the GMLK option on page 35).
3. The following points apply only if the object being closed is a *dynamic queue*:
   - For a dynamic queue, the options CODEL or COPURG can be specified regardless of the options specified on the corresponding MQOPEN call.
   - When a temporary dynamic queue is closed using the *HOBJ* handle returned by the MQOPEN call that created it, the queue is deleted (along with any messages that may still be on it) regardless of which of the valid options is specified in the *OPTS* parameter. This is true even if there are uncommitted MQGET, MQPUT, or MQPUT1 calls (issued using this or another handle) outstanding against the queue; any uncommitted updates that are lost *do not* cause the unit of work of which they are a part to fail.
   - When a dynamic queue is deleted, any MQGET requests with the GMWT option that are outstanding against the queue (using different *HOBJ* handles)

are canceled and reason code RC2052 is returned. See the GMWT option on page 28.

- After a dynamic queue (either temporary or permanent) has been deleted, any call (other than MQCLOSE) that attempts to reference the queue using another previously acquired *HOBJ* handle will fail with reason code RC2052.
- When an MQCLOSE call is issued to delete a permanent dynamic queue, using an *HOBJ* handle other than the one 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 (the alternate user identifier if OOALTU was specified) is authorized to delete the queue.

  No check is made when a temporary dynamic queue is deleted in this way, nor for a permanent dynamic queue if the handle specified is the one returned by the MQOPEN call that created the queue.

4. The following points apply only if the object being closed is a *distribution list*:
   - The only valid close option for a distribution list is CONONE; the call fails with reason code RC2046 or RC2045 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 *CMPCOD* 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 *CMPCOD* 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 CCFAIL, 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 *CMPCOD* and *REASON* parameters.
5. On OS/400, if the application was connected implicitly when the first MQOPEN call was issued, an implicit MQDISC occurs when the last MQCLOSE is issued.

RPG invocation (ILE)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                    CALLP     MQCLOSE(HCONN : HOBJ : OPTS :
C                                      CMPCOD : REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQCLOSE          PR                  EXTPROC('MQCLOSE')
D* Connection handle
D HCONN                         10I 0 VALUE
D* Object handle
D HOBJ                          10I 0
D* Options that control the action of MQCLOSE
D OPTS                          10I 0 VALUE
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CMPCOD
D REASON                        10I 0
```

RPG invocation (OPM)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                    CALL 'QMQM'
C* Call identifier
C                    PARM           CID     90
C* Connection handle
C                    PARM           HCONN   90
C* Object handle
C                    PARM           HOBJ    90
C* Options that control the action of QMQM
C                    PARM           OPTS    90
C* Completion code
C                    PARM           CMPCOD  90
C* Reason code qualifying CMPCOD
C                    PARM           REASON  90
```

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/400, this call need not be issued. Applications are connected automatically to the queue manager when they issue the first MQOPEN call. However, the MQCONN and MQDISC calls are still accepted from OS/400 applications.

```
MQCONN (QMNAME, HCONN, CMPCOD, REASON)
```

Parameters

QMNAME (48-byte character string) – input

Name of queue manager.

The name specified must be the name of a *connectable* queue manager. The name must not contain leading or embedded blanks, but may contain trailing blanks; the first null character and characters following it are treated as blanks. If the name consists entirely of blanks, the name of the *default* queue manager is used.

The queue managers to which it is possible to connect are determined by the environment:

- On OS/400, only the default queue manager is connectable.

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 on OS/400. However, OS/400 can act as an MQ server, to which MQ client applications can connect.

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 a 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 in this way implies that the application is not sensitive to which queue manager in the group the application is connected. This will not be suitable for certain types of application, for example those which need to get messages from a particular queue at a particular queue manager; such applications should not prefix the name with an asterisk. Use of queue-manager groups *is* suitable for applications that put messages, and/or get messages from temporary dynamic queues which they have created.

Note that if an asterisk is specified, the maximum length of the remainder of the name is 47 characters.

The length of this parameter is given by LNQMN. Queue-manager groups are not supported on OS/400.

HCONN (10-digit signed integer) – 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 within the environment concerned; the handle is not valid outside the unit of parallel processing from which the MQCONN call was issued.

- On OS/400, the scope of the handle is the job issuing the call.

On OS/400, the value returned is:

HCDEFH
Default connection handle.

CMPCOD (10-digit signed integer) – output
Completion code.

It is one of the following:

CCOK
Successful completion.

CCWARN
Warning (partial completion).

CCFAIL
: Call failed.

REASON (10-digit signed integer) – output
: Reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

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

If *CMPCOD* is CCWARN:

RC2002
: (2002, X'7D2') Application already connected.

If *CMPCOD* is CCFAIL:

RC2219
: (2219, X'8AB') MQI call reentered before previous call complete.

RC2009
: (2009, X'7D9') Connection to queue manager lost.

RC2018
: (2018, X'7E2') Connection handle not valid.

RC2035
: (2035, X'7F3') Not authorized for access.

RC2058
: (2058, X'80A') Queue manager name not valid or not known.

RC2059
: (2059, X'80B') Queue manager not available for connection.

RC2161
: (2161, X'871') Queue manager quiescing.

RC2162
: (2162, X'872') Queue manager shutting down.

RC2102
: (2102, X'836') Insufficient system resources available.

RC2063
: (2063, X'80F') Security error occurred.

RC2071
: (2071, X'817') Insufficient storage available.

RC2195
: (2195, X'893') Unexpected error occurred.

For more information on these reason codes, see Chapter 5, "Return codes" on page 275.

Usage notes

1. The queue manager to which connection is made using the MQCONN call is called the *local queue manager*.
2. Queues that belong to the local queue manager appear to the application as local queues. It is possible to put messages on and get messages from local queues.

   Queues belonging to remote queue managers appear as remote queues. It is possible to put messages on remote queues, but not possible to get messages from remote queues.

3. After a failure of the queue manager, this call must be reissued. The application program can periodically reissue MQCONN calls until it finds that the queue manager has been restarted. If an application is not sure whether or not it is connected to the queue manager, it can safely reissue an MQCONN call. If the application is already connected, the same handle from the previous MQCONN call is returned, together with a warning completion code and reason code RC2002.
4. Use the MQDISC call to disconnect from the queue manager.

RPG invocation (ILE)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQCONN(QMNAME : HCONN : CMPCOD :
C                                      REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQCONN           PR                  EXTPROC('MQCONN')
D* Name of queue manager
D QMNAME                        48A
D* Connection handle
D HCONN                         10I 0
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CMPCOD
D REASON                        10I 0
```

RPG invocation (OPM)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALL 'QMQM'
C* Call identifier
C                     PARM           CID     90
C* Name of queue manager
C                     PARM           QMNAME 48
C* Connection handle
C                     PARM           HCONN   90
C* Completion code
C                     PARM           CMPCOD  90
C* Reason code qualifying CMPCOD
C                     PARM           REASON  90
```

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 call.

On OS/400, this call need not be issued. See "MQCONN – Connect queue manager" on page 175 for more information.

```
MQDISC (HCONN, CMPCOD, REASON)
```

Parameters

HCONN (10-digit signed integer) – input/output

Connection handle.

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN call.

On OS/400, both the MQCONN and MQDISC calls can be omitted, and the following value used where *HCONN* would normally be specified:

HCDEFH
: 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:

HCUNUH
: Unusable connection handle.

CMPCOD (10-digit signed integer) – output

Completion code.

It is one of the following:

CCOK
: Successful completion.

CCWARN
: Warning (partial completion).

CCFAIL
: Call failed.

REASON (10-digit signed integer) – output

Reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

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

If *CMPCOD* is CCFAIL:

RC2219
: (2219, X'8AB') MQI call reentered before previous call complete.

RC2009
: (2009, X'7D9') Connection to queue manager lost.

RC2018
(2018, X'7E2') Connection handle not valid.
RC2058
(2058, X'80A') Queue manager name not valid or not known.
RC2059
(2059, X'80B') Queue manager not available for connection.
RC2162
(2162, X'872') Queue manager shutting down.
RC2102
(2102, X'836') Insufficient system resources available.
RC2071
(2071, X'817') Insufficient storage available.
RC2195
(2195, X'893') Unexpected error occurred.

For more information on these reason codes, see Chapter 5, “Return codes” on page 275.

Usage notes

1. If an MQDISC call is issued when the application still has objects open, these objects are implicitly closed, with the close options set to CONONE.
2. On OS/400, this call need not be used; see the MQCONN call for more details.

RPG invocation (ILE)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQDISC(HCONN : CMPCOD : REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQDISC           PR                    EXTPROC('MQDISC')
D* Connection handle
D HCONN                         10I 0
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CMPCOD
D REASON                        10I 0
```

RPG invocation (OPM)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALL 'QMQM'
C* Call identifier
C                     PARM           CID     90
C* Connection handle
C                     PARM           HCONN   90
C* Completion code
C                     PARM           CMPCOD  90
C* Reason code qualifying CMPCOD
C                     PARM           REASON  90
```

MQGET – Get message

The MQGET call retrieves a message from a local queue that has been opened using the MQOPEN call.

```
MQGET (HCONN, HOBJ, MSGDSC, GMO, BUFLEN, BUFFER, DATLEN, CMPCOD,
       REASON)
```

Parameters

HCONN (10-digit signed integer) – input
: Connection handle.

  This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN call.

  On OS/400, the MQCONN call can be omitted, and the following value specified for *HCONN*:

  HCDEFH
  : Default connection handle.

HOBJ (10-digit signed integer) – 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 “MQOPEN – Open object” on page 204 for details):

  OOINPS
  OOINPX
  OOINPQ
  OOBRW

MSGDSC (MQMD) – input/output
: Message descriptor.

  This structure describes the attributes of the message required, and the attributes of the message retrieved. See “MQMD – Message descriptor” on page 59 for details.

  If *BUFLEN* is less than the message length, *MSGDSC* is still filled in by the queue manager, whether or not GMATM is specified on the *GMO* parameter (see the *GMOPT* field on page 28 for more information).

  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 FMMDE in the *MDFMT* field in MQMD indicates that an MQMDE is present.

GMO (MQGMO) – input/output
: Options that control the action of MQGET.

  See “MQGMO – Get message options” on page 27 for details.

BUFLEN (10-digit signed integer) – 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 (GMATM 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* local queue attribute; see page 252.

BUFFER (1-byte bit string×*BUFLEN*) – output
Area to contain the message data.

If *BUFLEN* is less than the message length, as much of the message as possible is moved into *BUFFER*; this happens whether or not GMATM is specified on the *GMO* parameter (see the *GMOPT* field on page 28 for more information).

The character set and encoding of the data in *BUFFER* are given (respectively) by the *MDCSI* and *MDENC* fields returned in the *MSGDSC* 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 GMCONV option can be used with a user-written exit to perform the conversion of the message data (see page 37 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 ENNAT, respectively).

If the call fails, the contents of the buffer may still have changed.

DATLEN (10-digit signed integer) – output
Length of the message.

This is the length in bytes of the application data *in the message*. If this is greater than *BUFLEN*, only *BUFLEN* 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 *BUFLEN* is less than the message length, *DATLEN* is still filled in by the queue manager, whether or not GMATM is specified on the *GMO* parameter (see the *GMOPT* field on page 28 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 GMCONV option is specified, and the converted message data is too long to fit in *BUFFER*, the value returned for *DATLEN* 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 *DATLEN*.

- The value returned by the data-conversion exit, for application-defined formats.

CMPCOD (10-digit signed integer) – output
Completion code.

It is one of the following:

CCOK
Successful completion.
CCWARN
Warning (partial completion).
CCFAIL
Call failed.

REASON (10-digit signed integer) – output
Reason code qualifying *CMPCOD*.

The reason codes listed below are the ones that the queue manager can return for the *REASON* parameter. If the application specifies the GMCONV 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 *CMPCOD* is CCOK :

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

If *CMPCOD* is CCWARN:

RC2120
(2120, X'848') Converted message too big for application buffer.
RC2110
(2110, X'83E') Message format not valid.
RC2243
(2243, X'8C3') Message segments have differing CCSIDs.
RC2244
(2244, X'8C4') Message segments have differing encodings.
RC2209
(2209, X'8A1') No message locked.
RC2119
(2119, X'847') Application message data not converted.
RC2111
(2111, X'83F') Source coded character set identifier not valid.
RC2113
(2113, X'841') Packed-decimal encoding in message not recognized.
RC2114
(2114, X'842') Floating-point encoding in message not recognized.
RC2112
(2112, X'840') Source integer encoding not recognized.
RC2115
(2115, X'843') Target coded character set identifier not valid.

RC2117
: (2117, X'845') Packed-decimal encoding specified by receiver not recognized.

RC2118
: (2118, X'846') Floating-point encoding specified by receiver not recognized.

RC2116
: (2116, X'844') Target integer encoding not recognized.

RC2079
: (2079, X'81F') Truncated message returned (processing completed).

RC2080
: (2080, X'820') Truncated message returned (processing not completed).

If *CMPCOD* is CCFAIL:

RC2004
: (2004, X'7D4') Buffer parameter not valid.

RC2005
: (2005, X'7D5') Buffer length parameter not valid.

RC2219
: (2219, X'8AB') MQI call reentered before previous call complete.

RC2009
: (2009, X'7D9') Connection to queue manager lost.

RC2010
: (2010, X'7DA') Data length parameter not valid.

RC2016
: (2016, X'7E0') Gets inhibited for the queue.

RC2186
: (2186, X'88A') Get-message options structure not valid.

RC2018
: (2018, X'7E2') Connection handle not valid.

RC2019
: (2019, X'7E3') Object handle not valid.

RC2241
: (2241, X'8C1') Message group not complete.

RC2242
: (2242, X'8C2') Logical message not complete.

RC2259
: (2259, X'8D3') Inconsistent browse specification.

RC2245
: (2245, X'8C5') Inconsistent unit-of-work specification.

RC2246
: (2246, X'8C6') Message under cursor not valid for retrieval.

RC2247
: (2247, X'8C7') Match options not valid.

RC2026
: (2026, X'7EA') Message descriptor not valid.

RC2033
: (2033, X'7F1') No message available.

RC2034
: (2034, X'7F2') Browse cursor not positioned on message.

RC2036
(2036, X'7F4') Queue not open for browse.
RC2037
(2037, X'7F5') Queue not open for input.
RC2041
(2041, X'7F9') Object definition changed since opened.
RC2101
(2101, X'835') Object damaged.
RC2046
(2046, X'7FE') Options not valid or not consistent.
RC2052
(2052, X'804') Queue has been deleted.
RC2058
(2058, X'80A') Queue manager name not valid or not known.
RC2059
(2059, X'80B') Queue manager not available for connection.
RC2161
(2161, X'871') Queue manager quiescing.
RC2162
(2162, X'872') Queue manager shutting down.
RC2102
(2102, X'836') Insufficient system resources available.
RC2071
(2071, X'817') Insufficient storage available.
RC2024
(2024, X'7E8') No more messages can be handled within current unit of work.
RC2072
(2072, X'818') Syncpoint support not available.
RC2195
(2195, X'893') Unexpected error occurred.
RC2255
(2255, X'8CF') Unit of work not available for the queue manager to use.
RC2090
(2090, X'82A') Wait interval in MQGMO not valid.
RC2256
(2256, X'8D0') Wrong version of MQGMO supplied.
RC2257
(2257, X'8D1') Wrong version of MQMD supplied.

For more information on these reason codes, see Chapter 5, “Return codes” on page 275.

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 GMBRWF or GMBRWN option is specified on the *GMO* parameter (see the *GMOPT* field on page 28).

   If the GMLK 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 GMUNLK option is specified, a previously-locked message is unlocked. No message is retrieved in this case, and the *MSGDSC*, *BUFLEN*, *BUFFER* and *DATLEN* parameters are not checked or altered.

2. If an application puts a sequence of messages on the same queue, the order of those messages is preserved provided that all of the following are true:
   - The messages all have the same priority.
   - 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 OS/400, the application is the job.
   - All of the MQPUT calls are within the same unit of work, or none of them is within a unit of work.
   - The queue is local to the queue manager at which the MQPUT calls were made (but see note 2b).

   If these conditions are satisfied, the messages will be presented to the receiving application in the order in which they were sent, provided that:
   - The receiver does not deliberately change the order of retrieval, for example by specifying a particular *MDMID* or *MDCID*.
   - 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 *MDCID* fields in the messages in a sequence to a value that was unique to that sequence of messages.

   Notes:

   a. 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.

   b. For remote queues, the order of the messages is preserved if the configuration is such that there is only one path from the sender’s 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 cannot be guaranteed. Messages destined for remote queues can also become out of sequence if one or more of them is put to a dead-letter queue (for example, because the destination queue is temporarily full).

   If the required conditions are not met, applications can include their own sequencing information within the application message data, or establish a conversation scheme in which each message is acknowledged, and the acknowledgement received by the sender, before the next message is put.

3. Applications should test for the feedback code FBQUIT in the *MDFB* field of the *MSGDSC* parameter. If this value is found, the application should end. See the *MDFB* field on page 74 for more information.

4. If the queue identified by *HOBJ* was opened with the OOSAVA option, and the completion code from the MQGET call is CCOK or CCWARN, the context associated with the queue handle *HOBJ* is set to the context of the message that has been retrieved (unless the GMBRWF or GMBRWN option is set, in which case it is marked as not available). This context can be used on a subsequent MQPUT or MQPUT1 call (for example, when a message is forwarded to another queue). For more information on message context, see the *MQSeries Application Programming Guide*.

5. If the GMCONV option is included in the *GMO* 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 *MDFMT* field in the control information in the message identifies the structure of the application data, and the *MDCSI* and *MDENC* fields in the control information in the message specify its character-set identifier and encoding. The application issuing the MQGET call specifies in the *MDCSI* and *MDENC* fields in the *MSGDSC* parameter the character-set identifier and encoding to which the application message data should be converted. If the *MDCSI* and *MDENC* values in the control information in the message are identical to those in the *MSGDSC* parameter, no conversion is necessary.

   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 *MDFMT* 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:

     FMADMN
     FMCMD1
     FMCMD2
     FMDLH
     FMEVNT
     FMIMS
     FMIMVS
     FMMDE
     FMPCF
     FMRMH
     FMSTR
     FMTM
     FMXQH

   - The format name FMNONE 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 GMCONV is specified on the MQGET call for a message that has a format name of FMNONE, and the character set or encoding of the message differs from that specified in the *MSGDSC* parameter, the message is still returned in the *BUFFER* parameter (assuming no other errors), but the call completes with completion code CCWARN and reason code RC2110.

     FMNONE 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 queue-manager-defined format names supported in the future.

  See Appendix D, "Data-conversion" on page 385 for details of the data-conversion exit.

On return from MQGET, the following reason code indicates that the message was converted successfully:

RCNONE

The following reason code indicates that the message *may* have been converted successfully; the application should check the *MDCSI* and *MDENC* fields in the *MSGDSC* parameter to find out:

RC2079

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 D, "Data-conversion" on page 385.

6. For the built-in formats listed above, the queue manager may perform *default conversion* of character strings in the message when the GMCONV 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 CCOK, instead of completing with CCWARN and reason code RC2111 or RC2115.

   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 GMCONV.
     - 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 GMCONV option is not specified by the application on the MQGET call.

7. The *BUFFER* parameter shown in the RPG programming example is declared as a string; this restricts the maximum length of the parameter to 256 bytes. If a larger buffer is required, the parameter should be declared instead as a structure, or as a field in a physical file.

   Declaring the parameter as a structure increases the maximum length possible to 9999 bytes, while declaring the parameter as a field in a physical file increases the maximum length possible to approximately 32K bytes.

RPG invocation (ILE)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQGET(HCONN : HOBJ : MSGDSC : GMO :
C                                     BUFLEN : BUFFER : DATLEN :
C                                     CMPCOD : REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQGET            PR                  EXTPROC('MQGET')
D* Connection handle
D HCONN                         10I 0 VALUE
D* Object handle
D HOBJ                          10I 0 VALUE
D* Message descriptor
D MSGDSC                       364A
D* Options that control the action of MQGET
D GMO                           80A
D* Length in bytes of the BUFFER area
D BUFLEN                        10I 0 VALUE
D* Area to contain the message data
D BUFFER                          *   VALUE
D* Length of the message
D DATLEN                        10I 0
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CMPCOD
D REASON                        10I 0
```

RPG invocation (OPM)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALL 'QMQM'
C* Call identifier
C                     PARM           CID     90
C* Connection handle
C                     PARM           HCONN   90
C* Object handle
C                     PARM           HOBJ    90
C* Message descriptor
C                     PARM           MSGDSC
C* Options that control the action of QMQM
C                     PARM           GMO
C* Length in bytes of the BUFFER area
C                     PARM           BUFLEN  90
C* Area to contain the message data
C                     PARM           BUFFER  n
C* Length of the message
C                     PARM           DATLEN  90
C* Completion code
C                     PARM           CMPCOD  90
C* Reason code qualifying CMPCOD
C                     PARM           REASON  90
```

Declare the structure parameters as follows:

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7..
I* Message descriptor
IMSGDSC      DS
I/COPY CMQMDR
I* Options that control the action of QMQM
IGMO         DS
I/COPY CMQGMOR
```

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
- Process definition
- Queue manager

```
MQINQ (HCONN, HOBJ, SELCNT, SELS, IACNT, INTATR, CALEN, CHRATR, CMPCOD,
              REASON)
```

Parameters

HCONN (10-digit signed integer) – input
: Connection handle.

  This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN call.

  On OS/400, the MQCONN call can be omitted, and the following value specified for *HCONN*:

  HCDEFH
  : Default connection handle.

HOBJ (10-digit signed integer) – 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 OOINQ option.

SELCNT (10-digit signed integer) – input
: Count of selectors.

  This is the count of selectors that are supplied in the *SELS* array. It is the number of attributes that are to be returned. Zero is a valid value. The maximum number allowed is 256.

SELS (10-digit signed integer×*SELCNT*) – input
: Array of attribute selectors.

  This is an array of *SELCNT* 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 CCFAIL and reason code RC2067.

  In the special case of queues:

  - If the selector is not valid for queues of *any* type, the call fails with completion code CCFAIL and reason code RC2067.
  - If the selector is applicable *only* to queues of type or types other than that of the object, the call succeeds with completion code CCWARN and reason code RC2068.

Selectors can be specified in any order. Attribute values that correspond to integer attribute selectors (IA* selectors) are returned in *INTATR* in the same order in which these selectors occur in *SELS*. Attribute values that correspond to character attribute selectors (CA* selectors) are returned in *CHRATR* in the same order in which those selectors occur. IA* selectors can be interleaved with the CA* 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 IA* selectors reside within the range IAFRST through IALAST, and the CA* selectors within the range CAFRST through CALAST.

   For each range, the constants IALSTU and CALSTU define the highest value that the queue manager will accept.

2. If all of the IA* selectors occur first, the same element numbers can be used to address corresponding elements in the *SELS* and *INTATR* arrays.

For the CA* selectors in the following descriptions, the constant that defines the length in bytes of the resulting string in *CHRATR* is given in parentheses.

Selectors for queue manager

CACADX
: Automatic channel definition exit name (LNEXN).

CACMDQ
: System command input queue name (LNQN).

CADLQ
: Name of dead-letter queue (LNQN).

CADXQN
: Default transmission queue name (LNQN).

CAQMD
: Queue manager description (LNQMD).

CAQMN
: Name of local queue manager (LNQMN).

IAAUTE
: Control attribute for authority events.

IACAD
: Control attribute for automatic channel definition.

IACADE
: Control attribute for automatic channel definition events.

IACCSI
: Coded character set identifier.

IACMDL
: Command level supported by queue manager.

IADIST
: Distribution list support.

IAINHE
: Control attribute for inhibit events.

IALCLE
: Control attribute for local events.

IAMHND
: Maximum number of handles.

IAMLEN
: Maximum message length.

IAMPRI
: Maximum priority.

IAMUNC
: Maximum number of uncommitted messages within a unit of work.

IAPFME
: Control attribute for performance events.

IAPLAT
: Platform on which the queue manager resides.

IARMTE
: Control attribute for remote events.

IASSE
: Control attribute for start stop events.

IASYNC
: Syncpoint availability.

IATRGI
: Trigger interval.

Selectors for all types of queue

CAQD
: Queue description (LNQD).

CAQN
: Queue name (LNQN).

IADPER
: Default message persistence.

IADPRI
: Default message priority.

IAIPUT
: Whether put operations are allowed.

IAQTYP
: Queue type.

Selectors for local queues

CABRQN
: Excessive backout requeue name (LNQN).

CACRTD
: Queue creation date (LNCRTD).

CACRTT
: Queue creation time (LNCRTT).

CAINIQ
: Initiation queue name (LNQN).

CAPRON
: Name of process definition (LNPRON).

CASTGC
: Name of storage class (LNSTGC).

CATRGD
: Trigger data (LNTRGD).

IABTHR
: Backout threshold.

IACDEP
: Number of messages on queue.

IADINP
: Default open-for-input option.

IADEFT
: Queue definition type.

IADIST
: Distribution list support.

IAHGB
: Whether to harden backout count.

IAIGET
: Whether get operations are allowed.

IAMLEN
: Maximum message length.

IAMDEP
: Maximum number of messages allowed on queue.

IAMDS
: Whether message priority is relevant.

IAOIC
: Number of MQOPEN calls that have the queue open for input.

IAOOC
: Number of MQOPEN calls that have the queue open for output.

IAQDHE
: Control attribute for queue depth high events.

IAQDHL
: High limit for queue depth.

IAQDLE
: Control attribute for queue depth low events.

IAQDLL
: Low limit for queue depth.

IAQDME
: Control attribute for queue depth max events.

IAQSI
: Limit for queue service interval.

IAQSIE
: Control attribute for queue service interval events.

IARINT
: Queue retention interval.

IASHAR
: Whether queue can be shared.

IATRGC
: Trigger control.

IATRGD
: Trigger depth.

IATRGP
: Threshold message priority for triggers.

IATRGT
: Trigger type.

IAUSAG
: Usage.

Selectors for local definitions of remote queues

CARQMN
: Name of remote queue manager (LNQMN).

CARQN
Name of remote queue as known on remote queue manager (LNQN).

CAXQN
Transmission queue name (LNQN).

Selectors for alias queues

CABASQ
Name of queue that alias resolves to (LNQN).

IAIGET
Whether get operations are allowed.

Selectors for process definitions

CAAPPI
Application identifier (LNPROA).

CAENVD
Environment data (LNPROE).

CAPROD
Description of process definition (LNPROD).

CAPRON
Name of process definition (LNPRON).

CAUSRD
User data (LNPROU).

IAAPPT
Application type.

IACNT (10-digit signed integer) – input
Count of integer attributes.

This is the number of elements in the *INTATR* array. Zero is a valid value.

If this is at least the number of IA* selectors in the *SELS* parameter, all integer attributes requested are returned.

INTATR (10-digit signed integer×*IACNT*) – output
Array of integer attributes.

This is an array of *IACNT* integer attribute values.

Integer attribute values are returned in the same order as the IA* selectors in the *SELS* parameter. If the array contains more elements than the number of IA* 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 IAVNA is returned for the corresponding element in the *INTATR* array.

CALEN (10-digit signed integer) – input
Length of character attributes buffer.

This is the length in bytes of the *CHRATR* parameter.

This must be at least the sum of the lengths of the requested character attributes (see *SELS*). Zero is a valid value.

CHRATR (1-byte character string×*CALEN*) – 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 *CALEN* parameter.

Character attributes are returned in the same order as the CA* selectors in the *SELS* parameter. The length of each attribute string is fixed for each attribute (see *SELS*), 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 *CHRATR*.

CMPCOD (10-digit signed integer) – output
Completion code.

It is one of the following:

CCOK
Successful completion.
CCWARN
Warning (partial completion).
CCFAIL
Call failed.

REASON (10-digit signed integer) – output
Reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

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

If *CMPCOD* is CCWARN:

RC2008
(2008, X'7D8') Not enough space allowed for character attributes.
RC2022
(2022, X'7E6') Not enough space allowed for integer attributes.
RC2068
(2068, X'814') Selector not applicable to queue type.

If *CMPCOD* is CCFAIL:

RC2219
(2219, X'8AB') MQI call reentered before previous call complete.
RC2006
(2006, X'7D6') Length of character attributes not valid.
RC2007
(2007, X'7D7') Character attributes string not valid.
RC2009
(2009, X'7D9') Connection to queue manager lost.
RC2018
(2018, X'7E2') Connection handle not valid.
RC2019
(2019, X'7E3') Object handle not valid.
RC2021
(2021, X'7E5') Count of integer attributes not valid.

RC2023
: (2023, X'7E7') Integer attributes array not valid.

RC2038
: (2038, X'7F6') Queue not open for inquire.

RC2041
: (2041, X'7F9') Object definition changed since opened.

RC2101
: (2101, X'835') Object damaged.

RC2052
: (2052, X'804') Queue has been deleted.

RC2058
: (2058, X'80A') Queue manager name not valid or not known.

RC2059
: (2059, X'80B') Queue manager not available for connection.

RC2162
: (2162, X'872') Queue manager shutting down.

RC2102
: (2102, X'836') Insufficient system resources available.

RC2065
: (2065, X'811') Count of selectors not valid.

RC2067
: (2067, X'813') Attribute selector not valid.

RC2066
: (2066, X'812') Count of selectors too big.

RC2071
: (2071, X'817') Insufficient storage available.

RC2195
: (2195, X'893') Unexpected error occurred.

For more information on these reason codes, see Chapter 5, “Return codes” on page 275.

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, even for inquiring about its attributes, a dynamic queue is created. The attributes of the dynamic queue (except for *CreationDate*, *CreationTime*, and *DefinitionType*) are the same as those of the model queue at the time the dynamic queue is created. If you subsequently use the MQINQ call with the same object handle, the queue manager returns the attributes of the dynamic queue, not those of the model queue.
3. The attributes returned by the MQINQ call directed at an alias queue are those of the alias queue, not those of the base queue to which the alias resolves.
4. 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.
5. If more than one of the warning situations arise (see the *CMPCOD* parameter), the reason code returned is the *first* one in the following list that applies:
   a. RC2068

   b. RC2022
   c. RC2008

6. For more information about object attributes, see Chapter 4, Attributes of MQSeries objects.

7. For OPM RPG, the following differences apply:

   - The name of the call is `QMQM`, instead of MQINQ.
   - An additional integer parameter is present at the beginning of the parameter list. This parameter is the call identifier, which must have the value MQINQ.
   - All integer parameters on the call must be 9-digit decimal integers, instead of 10-digit signed binary integers.

RPG invocation (ILE)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQINQ(HCONN : HOBJ : SELCNT :
C                                     SELS(1) : IACNT : INTATR(1) :
C                                     CALEN : CHRATR : CMPCOD :
C                                     REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQINQ            PR                  EXTPROC('MQINQ')
D* Connection handle
D HCONN                         10I 0 VALUE
D* Object handle
D HOBJ                          10I 0 VALUE
D* Count of selectors
D SELCNT                        10I 0 VALUE
D* Array of attribute selectors
D SELS                          10I 0
D* Count of integer attributes
D IACNT                         10I 0 VALUE
D* Array of integer attributes
D INTATR                        10I 0
D* Length of character attributes buffer
D CALEN                         10I 0 VALUE
D* Character attributes
D CHRATR                          *   VALUE
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CMPCOD
D REASON                        10I 0
```

RPG invocation (OPM)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALL 'QMQM'
C* Call identifier
C                     PARM           CID     90
C* Connection handle
C                     PARM           HCONN   90
C* Object handle
C                     PARM           HOBJ    90
C* Count of selectors
C                     PARM           SELCNT  90
C* Array of attribute selectors
C                     PARM           SELS
C* Count of integer attributes
C                     PARM           IACNT   90
C* Array of integer attributes
C                     PARM           INTATR
C* Length of character attributes buffer
C                     PARM           CALEN   90
C* Character attributes
C                     PARM           CHRATR  n
C* Completion code
C                     PARM           CMPCOD  90
C* Reason code qualifying CMPCOD
C                     PARM           REASON  90
```

Declare the array parameters as follows:

```
E*..1....:....2....:....3....:....4....:....5....:....6....:....7..
E* Array of attribute selectors
E                    SELS        n  9 0
E* Array of integer attributes
E                    INTATR      n  9 0
```

MQOPEN – Open object

The MQOPEN call establishes access to an object. The following types of object are valid:

- Queue (including distribution lists)
- Process definition (not 16-bit Windows, 32-bit Windows)
- Queue manager

MQOPEN *(HCONN, OBJDSC, OPTS, HOBJ, CMPCOD, REASON)*

Parameters

HCONN (10-digit signed integer) – input
Connection handle.

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN call.

On OS/400, the MQCONN call can be omitted, and the following value specified for *HCONN*:

HCDEFH
Default connection handle.

OBJDSC (MQOD) – input/output
Object descriptor.

This is a structure that identifies the object to be opened; see "MQOD – Object descriptor" on page 110 for details.

If the *ODON* field in the *OBJDSC* 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 *OPTS* 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 *OBJDSC* parameter is replaced with the name of the dynamic queue created. The type of the dynamic queue is determined by the value of the *DefinitionType* attribute of the model queue (see page 249). For information about the close options applicable to dynamic queues, see the description of the MQCLOSE call.

OPTS (10-digit signed integer) – input
Options that control the action of MQOPEN.

At least one of the following options must be specified:

OOBRW
OOINP* (only one of these)
OOINQ
OOOUT
OOSET

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). Combinations that are not valid are noted; all other combinations are valid. Only options that are applicable to the type of object specified by *OBJDSC* are allowed (see Table 39 on page 209).

The following options control the operations that can be performed on an object:

OOINPQ
: 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 page 250 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.

OOINPS
: 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 OOINPS, but fails with reason code RC2042 if the queue is currently open with OOINPX.

  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.

OOINPX
: Open queue to get messages with exclusive access.

  The queue is opened for use with subsequent MQGET calls. The call fails with reason code RC2042 if the queue is currently open by this or another application for input of any type (OOINPS or OOINPX).

  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:

OOINPQ
OOINPS
OOINPX

- 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 QAGETI (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* local-queue attribute has the value QANSHR), 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 *ODMN* 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.

OOBRW
: Open queue to browse messages.

  The queue is opened for use with subsequent MQGET calls with one of the following options:

  GMBRWF
  GMBRWN
  GMBRWC

  This is allowed even if the queue is currently open for OOINPX. An MQOPEN call with the OOBRW option establishes a browse cursor, and positions it logically before the first message on the queue—see the *GMOPT* field on page 28 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 *ODMN* 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.

OOOUT
: 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 QAPUTI (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.

OOINQ
: 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 *ODMN* 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.

OOSET
: 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 *ODMN* 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.

The following options control the processing of message context:

OOSAVA
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 PMPASI and PMPASA options on page 128.

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 GMBRW* browse options does **not** have its context information saved (although the context fields in the *MSGDSC* 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 OOINP* options must be specified.

OOPASI
Allow identity context to be passed.

This allows the PMPASI option to be specified in the *PMO* 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 OOSAVA option. For more information on message context, see the *MQSeries Application Programming Guide*.

The OOOUT option must be specified.

This option is valid for all types of queue, including distribution lists.

OOPASA
Allow all context to be passed.

This allows the PMPASA option to be specified in the *PMO* 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 OOSAVA option. For more information on message context, see the *MQSeries Application Programming Guide*.

This option implies OOPASI, which need not therefore be specified. The OOOUT option must be specified.

This option is valid for all types of queue, including distribution lists.

OOSETI
Allow identity context to be set.

This allows the PMSETI option to be specified in the *PMO* parameter when a message is put on a queue; this gives the message the identity context information contained in the *MSGDSC* parameter specified on the MQPUT or MQPUT1 call. For more information on message context, see the *MQSeries Application Programming Guide*.

This option implies OOPASI, which need not therefore be specified. The OOOUT option must be specified.

This option is valid for all types of queue, including distribution lists.

OOSETA
Allow all context to be set.

This allows the PMSETA option to be specified in the *PMO* parameter when a message is put on a queue; this gives the message the identity and origin context information contained in the *MSGDSC* 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:

OOPASI
OOPASA
OOSETI

The OOOUT option must be specified.

This option is valid for all types of queue, including distribution lists.

The following options control authorization checking, and what happens when the queue manager is quiescing:

OOALTU
Validate with specified user identifier.

This indicates that the *ODAU* field in the *OBJDSC* parameter contains a user identifier that is to be used to validate this MQOPEN call. The call can succeed only if this *ODAU* is authorized to open the object 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 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.

OOFIQ
Fail if queue manager is quiescing.

This option forces the MQOPEN call to fail if the queue manager is in quiescing state.

This option is valid for all types of object.

Table 39. Valid MQOPEN options for each queue type

Option	Alias (see note 1)	Local	Model	Remote	Distribution list
OOINPQ	√	√	√	—	—
OOINPS	√	√	√	—	—
OOINPX	√	√	√	—	—
OOBRW	√	√	√	—	—
OOOUT	√	√	√	√	√
OOINQ	√	√	√	√ (see note 2)	—
OOSET	√	√	√	√ (see note 2)	—
OOSAVA	√	√	√	—	—
OOPASI	√	√	√	√	√
OOPASA	√	√	√	√	√
OOSETI	√	√	√	√	√
OOSETA	√	√	√	√	√
OOALTU	√	√	√	√	√
OOFIQ	√	√	√	√	√

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.

HOBJ (10-digit signed integer) – 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 within the environment concerned; the handle is not valid outside the unit of parallel processing from which the MQOPEN call was issued:

- On OS/400, the scope of the handle is the job issuing the call.

CMPCOD (10-digit signed integer) – output
Completion code.

It is one of the following:

CCOK
Successful completion.

CCWARN
Warning (partial completion).

CCFAIL
Call failed.

REASON (10-digit signed integer) – output
Reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

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

If *CMPCOD* is CCFAIL:

RC2001
(2001, X'7D1') Alias base queue not a valid type.
RC2219
(2219, X'8AB') MQI call reentered before previous call complete.
RC2009
(2009, X'7D9') Connection to queue manager lost.
RC2198
(2198, X'896') Default transmission queue not local.
RC2199
(2199, X'897') Default transmission queue usage error.
RC2011
(2011, X'7DB') Name of dynamic queue not valid.
RC2017
(2017, X'7E1') No more handles available.
RC2018
(2018, X'7E2') Connection handle not valid.
RC2019
(2019, X'7E3') Object handle not valid.
RC2194
(2194, X'892') Object name not valid for object type.
RC2035
(2035, X'7F3') Not authorized for access.
RC2100
(2100, X'834') Object already exists.
RC2101
(2101, X'835') Object damaged.
RC2042
(2042, X'7FA') Object already open with conflicting options.
RC2043
(2043, X'7FB') Object type not valid.
RC2044
(2044, X'7FC') Object descriptor structure not valid.
RC2045
(2045, X'7FD') Option not valid for object type.
RC2046
(2046, X'7FE') Options not valid or not consistent.
RC2052
(2052, X'804') Queue has been deleted.
RC2058
(2058, X'80A') Queue manager name not valid or not known.
RC2059
(2059, X'80B') Queue manager not available for connection.

RC2161
: (2161, X'871') Queue manager quiescing.

RC2162
: (2162, X'872') Queue manager shutting down.

RC2057
: (2057, X'809') Queue type not valid.

RC2184
: (2184, X'888') Remote queue name not valid.

RC2102
: (2102, X'836') Insufficient system resources available.

RC2063
: (2063, X'80F') Security error occurred.

RC2071
: (2071, X'817') Insufficient storage available.

RC2195
: (2195, X'893') Unexpected error occurred.

RC2082
: (2082, X'822') Unknown alias base queue.

RC2197
: (2197, X'895') Unknown default transmission queue.

RC2085
: (2085, X'825') Unknown object name.

RC2086
: (2086, X'826') Unknown object queue manager.

RC2087
: (2087, X'827') Unknown remote queue manager.

RC2196
: (2196, X'894') Unknown transmission queue.

RC2091
: (2091, X'82B') Transmission queue not local.

RC2092
: (2092, X'82C') Transmission queue with wrong usage.

For more information on these reason codes, see Chapter 5, “Return codes” on page 275.

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 description of the *OBJDSC* parameter on page 204.

     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.

   - 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. 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 remote queue-manager name and the name by which that queue is known at the remote queue manager
   - Resolution of the remote queue-manager name to the name of a transmission queue

   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 *OPTS* parameter on the corresponding MQOPEN.

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 (see Usage note 3), regardless of the open options used. This includes the following:
     - A change to the *BaseQName* of an alias queue that is open.
     - With one exception, 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, 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.

       The exception is 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 *RemoteQName* or *RemoteQMgrName* remote 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.
   - The *Shareability* local queue attribute, if there are two or more handles that are currently providing OOINPS 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.

- The *Usage* local 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 RC2041; 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.

6. 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.

7. A remote queue can be specified in one of two ways in the *OBJDSC* parameter of this call (see the *ODON* and *ODMN* fields on page 111):

   - By specifying for *ODON* the name of a local definition of the remote queue. In this case, *ODMN* refers to the local queue manager, and can be specified as blanks.

     The security validation performed by the local queue manager verifies that the application is authorized to open the local definition of the remote queue.

   - By specifying for *ODON* the name of the remote queue as known to the remote queue manager. In this case, *ODMN* is the name of the remote queue manager.

     The security validation performed by the local queue manager verifies that the application 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 application 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.

8. The following notes apply to the use of distribution lists.

   Distribution lists are supported in the following environments: AIX, DOS client, HP-UX, OS/2, OS/400, Sun Solaris, Windows client, Windows NT.

a. Fields in the MQOD structure must be set as follows when opening a distribution list:

- *ODVER* must be ODVER2.
- *ODOT* must be OTQ.
- *ODON* must be blank or the null string.
- *ODMN* must be blank or the null string.
- *ODREC* must be greater than zero.
- One of *ODORO* and *ODORP* must be zero and the other nonzero.
- No more than one of *ODRRO* and *ODRRP* can be nonzero.
- There must be *ODREC* object records, addressed by either *ODORO* or *ODORP*. The object records must be set to the names of the destination queues to be opened.
- If one of *ODRRO* and *ODRRP* is nonzero, there must be *ODREC* response records present. They are set by the queue manager if the call completes with reason code RC2136.

A version-2 MQOD can also be used to open a single queue that is not in a distribution list, by ensuring that *ODREC* is zero.

b. Only the following open options are valid in the *OPTS* parameter:

OOOUT
OOPAS*
OOSET*
OOALTU
OOFIQ

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 RC2057. 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 CCOK and RCNONE respectively; if every open fails because none of the queues exists, the parameters are set to CCFAIL and RC2085.

- 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 CCWARN if at least one open succeeded, and to CCFAIL if all failed.
  – The reason code parameter is set to RC2136.
  – 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 CONONE.

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.

   In a similar fashion, each destination that is opened successfully may have the value of its *OpenOutputCount* attribute incremented by one.

g. 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.

h. It is valid for a distribution list to contain only one destination.

9. An MQOPEN call with the OOBRW 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 GMMUC option.

   Multiple browse cursors can be active for a single application by issuing several MQOPEN requests for the same queue.

10. On OS/400, the first MQOPEN call performs an implicit MQCONN function, if MQCONN has not already been issued.

11. 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 *OBJDSC* parameter to open the queue.

RPG invocation (ILE)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQOPEN(HCONN : OBJDSC : OPTS :
C                                      HOBJ : CMPCOD : REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQOPEN           PR                  EXTPROC('MQOPEN')
D* Connection handle
D HCONN                         10I 0 VALUE
D* Object descriptor
D OBJDSC                       224A
D* Options that control the action of MQOPEN
D OPTS                          10I 0 VALUE
D* Object handle
D HOBJ                          10I 0
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CMPCOD
D REASON                        10I 0
```

RPG invocation (OPM)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALL 'QMQM'
C* Call identifier
C                     PARM           CID     90
C* Connection handle
C                     PARM           HCONN   90
C* Object descriptor
C                     PARM           OBJDSC
C* Options that control the action of QMQM
C                     PARM           OPTS    90
C* Object handle
C                     PARM           HOBJ    90
C* Completion code
C                     PARM           CMPCOD  90
C* Reason code qualifying CMPCOD
C                     PARM           REASON  90
```

Declare the structure parameters as follows:

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7..
I* Object descriptor
IOBJDSC      DS
I/COPY CMQODR
```

MQPUT – Put message

The MQPUT call puts a message on a queue or distribution list. The queue or distribution list must already be open.

```
MQPUT (HCONN, HOBJ, MSGDSC, PMO, BUFLEN, BUFFER, CMPCOD, REASON)
```

Parameters

HCONN (10-digit signed integer) – input

Connection handle.

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN call.

On OS/400, the MQCONN call can be omitted, and the following value specified for *HCONN*:

HCDEFH

Default connection handle.

HOBJ (10-digit signed integer) – 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 OOOUT option.

MSGDSC (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 "MQMD – Message descriptor" on page 59 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 *MDFMT* field in the MQMD must be set to FMMDE to indicate that an MQMDE is present. See the description of the MQMDE structure on page 104 for more details.

PMO (MQPMO) – input/output

Options that control the action of MQPUT.

See "MQPMO – Put message options" on page 120 for details.

BUFLEN (10-digit signed integer) – input

Length of the message in *BUFFER*.

Zero is valid, and indicates that the message contains no application data.

If the destination queue is a local queue, or resolves to a local queue, *BUFLEN* cannot exceed the smaller of the *MaxMsgLength* local-queue attribute and *MaxMsgLength* queue-manager attribute.

If the destination queue is a remote queue, or resolves to a remote queue, *BUFLEN* cannot exceed the smaller of the *MaxMsgLength* local-queue and queue-manager attributes for any of the following:

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

However, 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 LNMHD bytes be subtracted from the *MaxMsgLength* values of the transmission queues when determining the limit for *BUFLEN*.

Note: Only failure to comply with condition 1 can be diagnosed synchronously (with reason code RC2030 or RC2031) 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 (1-byte bit string×*BUFLEN*) – input
Message data.

This is a buffer containing the application data to be sent.

If *BUFFER* contains character and/or numeric data, the *MDCSI* and *MDENC* fields in the *MSGDSC* 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 ENNAT, respectively).

CMPCOD (10-digit signed integer) – output
Completion code.

It is one of the following:

CCOK
Successful completion.
CCWARN
Warning (partial completion).
CCFAIL
Call failed.

REASON (10-digit signed integer) – output
Reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

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

If *CMPCOD* is CCWARN:

RC2136
    (2136, X'858') Multiple reason codes returned.
RC2049
    (2049, X'801') Message Priority exceeds maximum value supported.
RC2104
    (2104, X'838') Report option(s) in message descriptor not recognized.

If *CMPCOD* is CCFAIL:

RC2004
    (2004, X'7D4') Buffer parameter not valid.
RC2005
    (2005, X'7D5') Buffer length parameter not valid.
RC2219
    (2219, X'8AB') MQI call reentered before previous call complete.
RC2009
    (2009, X'7D9') Connection to queue manager lost.
RC2097
    (2097, X'831') Queue handle referred to does not save context.
RC2098
    (2098, X'832') Context not available for queue handle referred to.
RC2135
    (2135, X'857') Distribution header structure not valid.
RC2013
    (2013, X'7DD') Expiry time not valid.
RC2014
    (2014, X'7DE') Feedback code not valid.
RC2258
    (2258, X'8D2') Group identifier not valid.
RC2018
    (2018, X'7E2') Connection handle not valid.
RC2019
    (2019, X'7E3') Object handle not valid.
RC2241
    (2241, X'8C1') Message group not complete.
RC2242
    (2242, X'8C2') Logical message not complete.
RC2185
    (2185, X'889') Inconsistent persistence specification.
RC2245
    (2245, X'8C5') Inconsistent unit-of-work specification.
RC2026
    (2026, X'7EA') Message descriptor not valid.
RC2248
    (2248, X'8C8') Message descriptor extension not valid.
RC2027
    (2027, X'7EB') Missing reply-to queue.
RC2249
    (2249, X'8C9') Message flags not valid.
RC2250
    (2250, X'8CA') Message sequence number not valid.
RC2030
    (2030, X'7EE') Message length greater than maximum for queue.

RC2029
: (2029, X'7ED') Message type in message descriptor not valid.

RC2136
: (2136, X'858') Multiple reason codes returned.

RC2039
: (2039, X'7F7') Queue not open for output.

RC2093
: (2093, X'82D') Queue not open for pass all context.

RC2094
: (2094, X'82E') Queue not open for pass identity context.

RC2095
: (2095, X'82F') Queue not open for set all context.

RC2096
: (2096, X'830') Queue not open for set identity context.

RC2041
: (2041, X'7F9') Object definition changed since opened.

RC2101
: (2101, X'835') Object damaged.

RC2251
: (2251, X'8CB') Message segment offset not valid.

RC2137
: (2137, X'859') Queue not opened successfully.

RC2046
: (2046, X'7FE') Options not valid or not consistent.

RC2252
: (2252, X'8CC') Original length not valid.

RC2047
: (2047, X'7FF') Persistence not valid.

RC2048
: (2048, X'800') Message on a temporary dynamic queue cannot be persistent.

RC2173
: (2173, X'87D') Put-message options structure not valid.

RC2158
: (2158, X'86E') Put message record flags not valid.

RC2050
: (2050, X'802') Message priority not valid.

RC2051
: (2051, X'803') Put calls inhibited for the queue.

RC2159
: (2159, X'86F') Put message records not valid.

RC2052
: (2052, X'804') Queue has been deleted.

RC2053
: (2053, X'805') Queue already contains maximum number of messages.

RC2058
: (2058, X'80A') Queue manager name not valid or not known.

RC2059
: (2059, X'80B') Queue manager not available for connection.

RC2161
: (2161, X'871') Queue manager quiescing.

RC2162
: (2162, X'872') Queue manager shutting down.

RC2056
: (2056, X'808') No space available on disk for queue.

RC2154
: (2154, X'86A') Number of records present not valid.

RC2061
: (2061, X'80D') Report options in message descriptor not valid.

RC2156
: (2156, X'86C') Response records not valid.

RC2102
: (2102, X'836') Insufficient system resources available.

RC2253
: (2253, X'8CD') Length of data in message segment is zero.

RC2071
: (2071, X'817') Insufficient storage available.

RC2024
: (2024, X'7E8') No more messages can be handled within current unit of work.

RC2072
: (2072, X'818') Syncpoint support not available.

RC2195
: (2195, X'893') Unexpected error occurred.

RC2255
: (2255, X'8CF') Unit of work not available for the queue manager to use.

RC2257
: (2257, X'8D1') Wrong version of MQMD supplied.

For more information on these reason codes, see Chapter 5, "Return codes" on page 275.

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 OOOUT 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. The following notes apply to the use of distribution lists.

   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 *PMREC* 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 *PMVER* field must be set to PMVER2.

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 *PMREC* 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 CCOK and RCNONE respectively; if every put fails because all of the queues are inhibited for puts, the parameters are set to CCFAIL and RC2051.

- 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 CCWARN if at least one put succeeded, and to CCFAIL if all failed.
  – The reason code parameter is set to RC2136.
  – 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 CCFAIL and RC2137; that destination is included in *PMIDC*.

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 on page 250).

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 (for example, because the application specified PRQDEF or PEQDEF), 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.

3. 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 *BUFLEN* parameter on the MQPUT or MQPUT1 call. A PCF message is a message that has one of the following format names:

  FMADMN
  FMEVNT
  FMPCF

- 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.)

4. The *BUFFER* parameter shown in the RPG programming example is declared as a string; this restricts the maximum length of the parameter to 256 bytes. If a larger buffer is required, the parameter should be declared instead as a structure, or as a field in a physical file.

   Declaring the parameter as a structure increases the maximum length possible to 9999 bytes, while declaring the parameter as a field in a physical file increases the maximum length possible to approximately 32K bytes.

RPG invocation (ILE)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQPUT(HCONN : HOBJ : MSGDSC : PMO :
C                                     BUFLEN : BUFFER : CMPCOD :
C                                     REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQPUT            PR                  EXTPROC('MQPUT')
D* Connection handle
D HCONN                         10I 0 VALUE
D* Object handle
D HOBJ                          10I 0 VALUE
D* Message descriptor
D MSGDSC                       364A
D* Options that control the action of MQPUT
D PMO                          176A
D* Length of the message in BUFFER
D BUFLEN                        10I 0 VALUE
D* Message data
D BUFFER                          *   VALUE
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CMPCOD
D REASON                        10I 0
```

RPG invocation (OPM)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALL 'QMQM'
C* Call identifier
C                     PARM           CID     90
C* Connection handle
C                     PARM           HCONN   90
C* Object handle
C                     PARM           HOBJ    90
C* Message descriptor
C                     PARM           MSGDSC
C* Options that control the action of QMQM
C                     PARM           PMO
C* Length of the message in BUFFER
C                     PARM           BUFLEN  90
C* Message data
C                     PARM           BUFFER  n
C* Completion code
C                     PARM           CMPCOD  90
C* Reason code qualifying CMPCOD
C                     PARM           REASON  90
```

Declare the structure parameters as follows:

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7..
I* Message descriptor
IMSGDSC      DS
I/COPY CMQMDR
I* Options that control the action of QMQM
IPMO         DS
I/COPY CMQPMOR
```

MQPUT1 – Put one message

The MQPUT1 call puts one message on a queue. The queue need not be open.

```
MQPUT1 (HCONN, OBJDSC, MSGDSC, PMO, BUFLEN, BUFFER, CMPCOD, REASON)
```

Parameters

HCONN (10-digit signed integer) – input
: Connection handle.

  This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN call.

  On OS/400, the MQCONN call can be omitted, and the following value specified for *HCONN*:

  HCDEFH
  : Default connection handle.

OBJDSC (MQOD) – input
: Object descriptor.

  This is a structure which identifies the queue to which the message is added. See "MQOD – Object descriptor" on page 110 for details.

  The application must be authorized to open the queue for output. The queue must **not** be a model queue.

MSGDSC (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 "MQMD – Message descriptor" on page 59 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 *MDFMT* field in the MQMD must be set to FMMDE to indicate that an MQMDE is present. See the description of the MQMDE structure on page 104 for more details.

PMO (MQPMO) – input/output
: Options that control the action of MQPUT1.

  See "MQPMO – Put message options" on page 120 for details.

BUFLEN (10-digit signed integer) – input
: Length of the message in *BUFFER*.

  Zero is valid, and indicates that the message contains no application data.

  If the destination queue is a local queue, or resolves to a local queue, *BUFLEN* cannot exceed the smaller of the *MaxMsgLength* local-queue attribute and *MaxMsgLength* queue-manager attribute.

If the destination queue is a remote queue, or resolves to a remote queue, *BUFLEN* cannot exceed the smaller of the *MaxMsgLength* local-queue and queue-manager attributes for any of the following:

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

However, 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 LNMHD bytes be subtracted from the *MaxMsgLength* values of the transmission queues when determining the limit for *BUFLEN*.

Note: Only failure to comply with condition 1 can be diagnosed synchronously (with reason code RC2030 or RC2031) when the message is put. If conditions 2 or 3 are not satisfied, the message will be 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 (1-byte bit string×*BUFLEN*) – input
Message data.

This is a buffer containing the application message data to be sent.

If *BUFFER* contains character and/or numeric data, the *MDCSI* and *MDENC* fields in the *MSGDSC* 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 ENNAT, respectively).

CMPCOD (10-digit signed integer) – output
Completion code.

It is one of the following:

CCOK
Successful completion.

CCWARN
Warning (partial completion).

CCFAIL
Call failed.

REASON (10-digit signed integer) – output
Reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

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

If *CMPCOD* is CCWARN:

RC2136
(2136, X'858') Multiple reason codes returned.
RC2241
(2241, X'8C1') Message group not complete.
RC2242
(2242, X'8C2') Logical message not complete.
RC2049
(2049, X'801') Message Priority exceeds maximum value supported.
RC2104
(2104, X'838') Report option(s) in message descriptor not recognized.

If *CMPCOD* is CCFAIL:

RC2001
(2001, X'7D1') Alias base queue not a valid type.
RC2004
(2004, X'7D4') Buffer parameter not valid.
RC2005
(2005, X'7D5') Buffer length parameter not valid.
RC2219
(2219, X'8AB') MQI call reentered before previous call complete.
RC2009
(2009, X'7D9') Connection to queue manager lost.
RC2097
(2097, X'831') Queue handle referred to does not save context.
RC2098
(2098, X'832') Context not available for queue handle referred to.
RC2198
(2198, X'896') Default transmission queue not local.
RC2199
(2199, X'897') Default transmission queue usage error.
RC2135
(2135, X'857') Distribution header structure not valid.
RC2013
(2013, X'7DD') Expiry time not valid.
RC2014
(2014, X'7DE') Feedback code not valid.
RC2258
(2258, X'8D2') Group identifier not valid.
RC2017
(2017, X'7E1') No more handles available.
RC2018
(2018, X'7E2') Connection handle not valid.
RC2026
(2026, X'7EA') Message descriptor not valid.
RC2248
(2248, X'8C8') Message descriptor extension not valid.
RC2027
(2027, X'7EB') Missing reply-to queue.

RC2249
: (2249, X'8C9') Message flags not valid.

RC2250
: (2250, X'8CA') Message sequence number not valid.

RC2030
: (2030, X'7EE') Message length greater than maximum for queue.

RC2029
: (2029, X'7ED') Message type in message descriptor not valid.

RC2136
: (2136, X'858') Multiple reason codes returned.

RC2035
: (2035, X'7F3') Not authorized for access.

RC2101
: (2101, X'835') Object damaged.

RC2042
: (2042, X'7FA') Object already open with conflicting options.

RC2155
: (2155, X'86B') Object records not valid.

RC2043
: (2043, X'7FB') Object type not valid.

RC2044
: (2044, X'7FC') Object descriptor structure not valid.

RC2251
: (2251, X'8CB') Message segment offset not valid.

RC2046
: (2046, X'7FE') Options not valid or not consistent.

RC2252
: (2252, X'8CC') Original length not valid.

RC2047
: (2047, X'7FF') Persistence not valid.

RC2048
: (2048, X'800') Message on a temporary dynamic queue cannot be persistent.

RC2173
: (2173, X'87D') Put-message options structure not valid.

RC2158
: (2158, X'86E') Put message record flags not valid.

RC2050
: (2050, X'802') Message priority not valid.

RC2051
: (2051, X'803') Put calls inhibited for the queue.

RC2159
: (2159, X'86F') Put message records not valid.

RC2052
: (2052, X'804') Queue has been deleted.

RC2053
: (2053, X'805') Queue already contains maximum number of messages.

RC2058
: (2058, X'80A') Queue manager name not valid or not known.

RC2059
: (2059, X'80B') Queue manager not available for connection.

RC2161
: (2161, X'871') Queue manager quiescing.

RC2162
: (2162, X'872') Queue manager shutting down.

RC2056
: (2056, X'808') No space available on disk for queue.

RC2057
: (2057, X'809') Queue type not valid.

RC2154
: (2154, X'86A') Number of records present not valid.

RC2184
: (2184, X'888') Remote queue name not valid.

RC2061
: (2061, X'80D') Report options in message descriptor not valid.

RC2102
: (2102, X'836') Insufficient system resources available.

RC2156
: (2156, X'86C') Response records not valid.

RC2063
: (2063, X'80F') Security error occurred.

RC2253
: (2253, X'8CD') Length of data in message segment is zero.

RC2071
: (2071, X'817') Insufficient storage available.

RC2024
: (2024, X'7E8') No more messages can be handled within current unit of work.

RC2072
: (2072, X'818') Syncpoint support not available.

RC2195
: (2195, X'893') Unexpected error occurred.

RC2082
: (2082, X'822') Unknown alias base queue.

RC2197
: (2197, X'895') Unknown default transmission queue.

RC2085
: (2085, X'825') Unknown object name.

RC2086
: (2086, X'826') Unknown object queue manager.

RC2087
: (2087, X'827') Unknown remote queue manager.

RC2196
: (2196, X'894') Unknown transmission queue.

RC2255
: (2255, X'8CF') Unit of work not available for the queue manager to use.

RC2257
: (2257, X'8D1') Wrong version of MQMD supplied.

RC2091
: (2091, X'82B') Transmission queue not local.

RC2092
: (2092, X'82C') Transmission queue with wrong usage.

For more information on these reason codes, see Chapter 5, “Return codes” on page 275.

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 OOOUT 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. The MQPUT1 call can be used to put messages to distribution lists. For general information about this, see the usage notes for the MQOPEN and MQPUT calls.

   The following differences apply when using the MQPUT1 call:
   1. 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.
   2. The reason code RC2137 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 CCWARN, 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 RC2136.
   3. When the MQPUT1 call is used to put a message to a distribution list, the call counts as only one handle when checking whether the application has exceeded the permitted maximum number of handles (see the *MaxHandles* queue-manager attribute). This is true regardless of the number of destinations in the distribution list.

      In contrast, when the MQOPEN call is used, each destination in the distribution list counts as a separate handle.
3. 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 the usage notes for the MQPUT call.
4. If more than one of the warning situations arise (see the *CMPCOD* parameter), the reason code returned is the *first* one in the following list that applies:
   1. RC2136
   2. RC2242

   c. RC2241
   d. RC2049 or RC2104

5. The *BUFFER* parameter shown in the RPG programming example is declared as a string; this restricts the maximum length of the parameter to 256 bytes. If a larger buffer is required, the parameter should be declared instead as a structure, or as a field in a physical file.

   Declaring the parameter as a structure increases the maximum length possible to 9999 bytes, while declaring the parameter as a field in a physical file increases the maximum length possible to approximately 32K bytes.

RPG invocation (ILE)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQPUT1(HCONN : OBJDSC : MSGDSC :
C                                      PMO : BUFLEN : BUFFER :
C                                      CMPCOD : REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQPUT1           PR                  EXTPROC('MQPUT1')
D* Connection handle
D HCONN                         10I 0 VALUE
D* Object descriptor
D OBJDSC                       224A
D* Message descriptor
D MSGDSC                       364A
D* Options that control the action of MQPUT1
D PMO                          176A
D* Length of the message in BUFFER
D BUFLEN                        10I 0 VALUE
D* Message data
D BUFFER                          *   VALUE
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CMPCOD
D REASON                        10I 0
```

RPG invocation (OPM)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALL 'QMQM'
C* Call identifier
C                     PARM           CID     90
C* Connection handle
C                     PARM           HCONN   90
C* Object descriptor
C                     PARM           OBJDSC
C* Message descriptor
C                     PARM           MSGDSC
C* Options that control the action of QMQM
C                     PARM           PMO
C* Length of the message in BUFFER
C                     PARM           BUFLEN  90
C* Message data
C                     PARM           BUFFER  n
C* Completion code
C                     PARM           CMPCOD  90
C* Reason code qualifying CMPCOD
C                     PARM           REASON  90
```

Declare the structure parameters as follows:

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7..
I* Object descriptor
IOBJDSC      DS
I/COPY CMQODR
I* Message descriptor
IMSGDSC      DS
I/COPY CMQMDR
I* Options that control the action of QMQM
IPMO         DS
I/COPY CMQPMOR
```

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.

MQSET *(HCONN, HOBJ, SELCNT, SELS, IACNT, INTATR, CALEN, CHRATR, CMPCOD, REASON)*

Parameters

HCONN (10-digit signed integer) – input
Connection handle.

This handle represents the connection to the queue manager. The value of *HCONN* was returned by a previous MQCONN call.

On OS/400, the MQCONN call can be omitted, and the following value specified for *HCONN*:

HCDEFH
Default connection handle.

HOBJ (10-digit signed integer) – 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 OOSET option.

SELCNT (10-digit signed integer) – input
Count of selectors.

This is the count of selectors that are supplied in the *SELS* array. It is the number of attributes that are to be set. Zero is a valid value. The maximum number allowed is 256.

SELS (10-digit signed integer×*SELCNT*) – input
Array of attribute selectors.

This is an array of *SELCNT* 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 IA* and CA* values are allowed; these values are listed below.

Selectors can be specified in any order. Attribute values that correspond to integer attribute selectors (IA* selectors) must be specified in *INTATR* in the same order in which these selectors occur in *SELS*. Attribute values that correspond to character attribute selectors (CA* selectors) must be specified in *CHRATR* in the same order in which those selectors occur. IA* selectors can be interleaved with the CA* 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 IA* selectors reside within the range IAFRST through IALAST, and the CA* selectors within the range CAFRST through CALAST.

   For each range, the constants IALSTU and CALSTU define the highest value that the queue manager will accept.

2. If all the IA* selectors occur first, the same element numbers can be used to address corresponding elements in the *SELS* and *INTATR* arrays.

For the CA* selectors in the following descriptions, the constant that defines the length in bytes of the string that is required in *CHRATR* is given in parentheses.

Selectors for all types of queue

IAIPUT
: Whether put operations are allowed.

Selectors for local queues

CATRGD
: Trigger data (LNTRGD).

IADIST
: Distribution list support.

IAIGET
: Whether get operations are allowed.

IATRGC
: Trigger control.

IATRGD
: Trigger depth.

IATRGP
: Threshold message priority for triggers.

IATRGT
: Trigger type.

Selectors for alias queues

IAIGET
: Whether get operations are allowed.

No other attributes can be set using this call.

IACNT (10-digit signed integer) – input
Count of integer attributes.

This is the number of elements in the *INTATR* array, and must be at least the number of IA* selectors in the *SELS* parameter. Zero is a valid value if there are none.

INTATR (10-digit signed integer×*IACNT*) – input
Array of integer attributes.

This is an array of *IACNT* integer attribute values. These attribute values must be in the same order as the IA* selectors in the *SELS* array.

CALEN (10-digit signed integer) – input
Length of character attributes buffer.

This is the length in bytes of the *CHRATR* parameter, and must be at least the sum of the lengths of the character attributes specified in the *SELS* array. Zero is a valid value if there are no CA* selectors in *SELS*.

CHRATR (1-byte character string×*CALEN*) – input
Character attributes.

This is the buffer containing the character attribute values, concatenated together. The length of the buffer is given by the *CALEN* parameter.

The characters attributes must be specified in the same order as the CA* selectors in the *SELS* array. The length of each character attribute is fixed (see *SELS*). If the value to be set for an attribute contains fewer nonblank characters than the defined length of the attribute, the value in *CHRATR* must be padded to the right with blanks to make the attribute value match the defined length of the attribute.

CMPCOD (10-digit signed integer) – output
Completion code.

It is one of the following:

CCOK
Successful completion.

CCFAIL
Call failed.

REASON (10-digit signed integer) – output
Reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

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

If *CMPCOD* is CCFAIL:

RC2219
(2219, X'8AB') MQI call reentered before previous call complete.

RC2006
(2006, X'7D6') Length of character attributes not valid.

RC2007
(2007, X'7D7') Character attributes string not valid.

RC2009
(2009, X'7D9') Connection to queue manager lost.

RC2018
(2018, X'7E2') Connection handle not valid.

RC2019
(2019, X'7E3') Object handle not valid.

RC2020
(2020, X'7E4') Value for inhibit-get or inhibit-put queue attribute not valid.

RC2021
(2021, X'7E5') Count of integer attributes not valid.

RC2023
(2023, X'7E7') Integer attributes array not valid.

RC2040
: (2040, X'7F8') Queue not open for set.

RC2041
: (2041, X'7F9') Object definition changed since opened.

RC2101
: (2101, X'835') Object damaged.

RC2052
: (2052, X'804') Queue has been deleted.

RC2058
: (2058, X'80A') Queue manager name not valid or not known.

RC2059
: (2059, X'80B') Queue manager not available for connection.

RC2162
: (2162, X'872') Queue manager shutting down.

RC2102
: (2102, X'836') Insufficient system resources available.

RC2065
: (2065, X'811') Count of selectors not valid.

RC2067
: (2067, X'813') Attribute selector not valid.

RC2066
: (2066, X'812') Count of selectors too big.

RC2071
: (2071, X'817') Insufficient storage available.

RC2075
: (2075, X'81B') Value for trigger-control attribute not valid.

RC2076
: (2076, X'81C') Value for trigger-depth attribute not valid.

RC2077
: (2077, X'81D') Value for trigger-message-priority attribute not valid.

RC2078
: (2078, X'81E') Value for trigger-type attribute not valid.

RC2195
: (2195, X'893') Unexpected error occurred.

For more information on these reason codes, see Chapter 5, “Return codes” on page 275.

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 “MQINQ – Inquire about object attributes” on page 194 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 OOSET option, you can use the MQSET call to set the attributes of the dynamic queue that is created by the MQOPEN call.

5. For more information about object attributes, see Chapter 4, Attributes of MQSeries objects.

6. For OPM RPG, the following differences apply:

   - The name of the call is `QMQM`, instead of MQSET.
   - An additional integer parameter is present at the beginning of the parameter list. This parameter is the call identifier, which must have the value MQSET.
   - All integer parameters on the call must be 9-digit decimal integers, instead of 10-digit signed binary integers.

RPG invocation (ILE)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALLP     MQSET(HCONN : HOBJ : SELCNT :
C                                     SELS(1) : IACNT : INTATR(1) :
C                                     CALEN : CHRATR : CMPCOD :
C                                     REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQSET            PR                  EXTPROC('MQSET')
D* Connection handle
D HCONN                         10I 0 VALUE
D* Object handle
D HOBJ                          10I 0 VALUE
D* Count of selectors
D SELCNT                        10I 0 VALUE
D* Array of attribute selectors
D SELS                          10I 0
D* Count of integer attributes
D IACNT                         10I 0 VALUE
D* Array of integer attributes
D INTATR                        10I 0
D* Length of character attributes buffer
D CALEN                         10I 0 VALUE
D* Character attributes
D CHRATR                          *   VALUE
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CMPCOD
D REASON                        10I 0
```

RPG invocation (OPM)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                     CALL 'QMQM'
C* Call identifier
C                     PARM           CID     90
C* Connection handle
C                     PARM           HCONN   90
C* Object handle
C                     PARM           HOBJ    90
C* Count of selectors
C                     PARM           SELCNT  90
C* Array of attribute selectors
C                     PARM           SELS
C* Count of integer attributes
C                     PARM           IACNT   90
C* Array of integer attributes
C                     PARM           INTATR
C* Length of character attributes buffer
C                     PARM           CALEN   90
C* Character attributes
C                     PARM           CHRATR  n
C* Completion code
C                     PARM           CMPCOD  90
C* Reason code qualifying CMPCOD
C                     PARM           REASON  90
```

Declare the array parameters as follows:

```
E*..1....:....2....:....3....:....4....:....5....:....6....:....7..
E* Array of attribute selectors
E                    SELS        n  9 0
E* Array of integer attributes
E                    INTATR      n  9 0
```

Chapter 4. Attributes of MQSeries objects

MQSeries objects consist of:

- Channels
- Queues
- Queue managers
- Namelists (MVS/ESA only)
- Processes
- Storage Classes (MVS/ESA only)

This chapter describes the attributes (or properties) of MQSeries objects that are accessible through the API, which are queues, queue managers, and processes.

The attributes are grouped according to the type of object to which they apply; see:

- “Attributes for all queues”
- “Attributes for local queues” on page 246
- “Attributes for local definitions of remote queues” on page 260
- “Attributes for alias queues” on page 262
- “Attributes for process definitions” on page 262
- “Attributes for the queue-manager” on page 264

Within each section, the attributes are listed in alphabetic order.

Note: The names of the attributes of objects are shown in this book in the form that you use them with the MQINQ and MQSET calls. When you use MQSeries commands to define, alter, or display the attributes, you use the keywords shown in the descriptions of the commands in the *MQSeries Command Reference*.

Attributes for all queues

The following table summarizes the attributes that are common to all queue types (except where noted). The attributes are described in alphabetic order.

Table 40. Attributes for all queues

Attribute	Description	Page
DefPersistence	Default message persistence	244
DefPriority	Default message priority	244
InhibitGet	Controls whether get operations for the queue are allowed	245
InhibitPut	Controls whether put operations for the queue are allowed	245
QDesc	Queue description	246
QName	Queue name	246
QType	Queue type	246

© Copyright IBM Corp. 1994, 1998

DefPersistence (10-digit signed integer)
Default message persistence.

This is the default persistence for messages on a queue. This applies if PEQDEF 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 put operation (even if this is a queue-manager alias).

The value is one of the following:

PEPER
Message is persistent.

The message survives restarts of the queue manager. Because temporary dynamic queues *do not* survive restarts of the queue manager, persistent messages cannot be put on temporary dynamic queues; persistent messages can however be put on permanent dynamic queues, and predefined queues.

PENPER
Message is not persistent.

The message does not survive restarts of the queue manager. This applies even if an intact copy of the message is found on auxiliary storage during the restart procedure.

Both persistent and nonpersistent messages can exist on the same queue.

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

DefPriority (10-digit signed integer)
Default message priority

This is the default priority for messages on the queue. This applies if PRQDEF 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 (even if this is a queue-manager alias).

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 MSPRIO, the logical position at which a message is placed on the queue is dependent on the value of the *MDPRI* field in the message descriptor.
- If the *MsgDeliverySequence* attribute is MSFIFO, 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 *MDPRI* field in the message descriptor. However, the *MDPRI* field retains the value specified by the application which put the msg. See the *MsgDeliverySequence* attribute described in "Attributes for local queues" on page 246 for more information.

Priorities are in the range zero (lowest) through *MaxPriority* (highest); see the *MaxPriority* attribute described in "Attributes for the queue-manager" on page 264.

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

InhibitGet (10-digit signed integer)
Controls whether get operations for this queue are allowed.

This attribute applies only to local, model, and alias queues.

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:

QAGETI
Get operations are inhibited.

MQGET calls fail with reason code RC2016. This includes MQGET calls that specify GMBRWF or GMBRWN.

Note: If an MQGET call operating within a unit of work completes successfully, changing the value of the *InhibitGet* attribute subsequently to QAGETI does not prevent the unit of work being committed.

QAGETA
Get operations are allowed.

To determine the value of this attribute, use the IAIGET selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

InhibitPut (10-digit signed integer)
Controls whether put operations for this queue are allowed.

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:

QAPUTI
Put operations are inhibited.

MQPUT and MQPUT1 calls fail with reason code RC2051.

Note: If an MQPUT call operating within a unit of work completes successfully, changing the value of the *InhibitPut* attribute subsequently to QAPUTI does not prevent the unit of work being committed.

QAPUTA
Put operations are allowed.

To determine the value of this attribute, use the IAIPUT selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

QDesc (64-byte character string)
Queue 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 CAQD selector with the MQINQ call. The length of this attribute is given by LNQD.

QName (48-byte character string)
Queue name.

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 QTLOC queue and a QTALS queue cannot have the same name.

To determine the value of this attribute, use the CAQN selector with the MQINQ call. The length of this attribute is given by LNQN.

QType (10-digit signed integer)
Queue type.

This attribute has one of the following values:

QTALS
Alias queue definition.
QTLOC
Local queue.
QTMOD
Model queue definition.
QTREM
Local definition of a remote queue.

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

Attributes for local queues

The following table summarizes the attributes that are specific to local queues and model queues (except where noted). The attributes are described in alphabetic order.

Table 41 (Page 1 of 2). Attributes for local and model queues

Attribute	Description	Page
BackoutRequeueQName	Excessive backout requeue queue name	247
BackoutThreshold	Backout threshold	248

Table 41 (Page 2 of 2). Attributes for local and model queues

Attribute	Description	Page
CreationDate	Date the queue was created	248
CreationTime	Time the queue was created	248
CurrentQDepth	Current queue depth	248
DefinitionType	Queue definition type	249
DefInputOpenOption	Default input open option	250
DistLists	Distribution list support	250
HardenGetBackout	Whether to maintain an accurate backout count	251
InitiationQName	Name of initiation queue	252
MaxMsgLength	Maximum message length in bytes	252
MaxQDepth	Maximum queue depth	252
MsgDeliverySequence	Message delivery sequence	253
OpenInputCount	Number of opens for input	254
OpenOutputCount	Number of opens for output	254
ProcessName	Process name	254
QDepthHighEvent	Controls whether Queue Depth High events are generated	254
QDepthHighLimit	High limit for queue depth	255
QDepthLowEvent	Controls whether Queue Depth Low events are generated	255
QDepthLowLimit	Low limit for queue depth	255
QDepthMaxEvent	Controls whether Queue Full events are generated	256
QServiceInterval	Target for queue service interval	256
QServiceIntervalEvent	Controls whether Service Interval High or Service Interval OK events are generated	256
RetentionInterval	Retention interval	257
Shareability	Queue shareability	257
TriggerControl	Trigger control	257
TriggerData	Trigger data	258
TriggerDepth	Trigger depth	258
TriggerMsgPriority	Threshold message priority for triggers	258
TriggerType	Trigger type	259
Usage	Queue usage	259

The attributes are described in alphabetic order.

BackoutRequeueQName (48-byte character string)

Excessive backout requeue queue name.

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 CABRQN selector with the MQINQ call. The length of this attribute is given by LNQN.

BackoutThreshold (10-digit signed integer)
Backout threshold.

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 IABTHR selector with the MQINQ call.

CreationDate (12-byte character string)
Date this queue was created.

The format is

```
YYYY-MM-DD
```

with 2 bytes of blank padding to the right to make the length 12 bytes. For example:

```
1992-09-23bb
```

is 23 September 1992 ("bb" represents 2 blank characters).

On OS/400, the creation date of a queue may differ from that of the underlying operating system entity (file or userspace) that represents the queue.

To determine the value of this attribute, use the CACRTD selector with the MQINQ call. The length of this attribute is given by LNCRTD.

CreationTime (8-byte character string)
Time this queue was created.

The format is

```
HH.MM.SS
```

using the 24-hour clock, with a leading zero if the hour is less than 10. For example:

```
21.10.20
```

This is an 8-character string. The time is local time.

- On OS/400, the creation time of a queue may differ from that of the underlying operating system entity (file or userspace) that represents the queue.

To determine the value of this attribute, use the CACRTT selector with the MQINQ call. The length of this attribute is given by LNCRTT.

CurrentQDepth (10-digit signed integer)
Current queue depth.

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 *MDEXP* field described in “MQMD – Message descriptor” on page 59.

The value of this attribute fluctuates as the queue manager operates.

This attribute does not apply to model queues, but it does apply to the dynamically-defined queues created from the model queue definitions using the MQOPEN call.

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

DefinitionType (10-digit signed integer)
Queue definition type.

This indicates how the queue was defined. It is one of the following:

QDPRE
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 command, and can be deleted only by using the DELETE command. Predefined queues cannot be created from model queues.

Commands can be issued either by an operator, or by an authorized application sending a command message to the command input queue (see the *CommandInputQName* attribute described in “Attributes for the queue-manager” on page 264).

QDPERM
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. The model queue definition has the value QDPERM for the *DefinitionType* attribute. This type of queue can be deleted using the MQCLOSE call. See “MQCLOSE – Close object” on page 169 for more details.

QDTEMP
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. The model queue definition has the value QDTEMP 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.

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 IADEFT selector with the MQINQ call.

DefInputOpenOption (10-digit signed integer)
Default input open option.

This is the default way in which the queue should be opened for input. It applies if the OOINPQ option is specified on the MQOPEN call when the queue is opened. It is one of the following:

OOINPX
Open queue to get messages with exclusive access.

The queue is opened for use with subsequent MQGET calls. The call fails with reason code RC2042 if the queue is currently open by this or another application for input of any type (OOINPS or OOINPX).

OOINPS
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 OOINPS, but fails with reason code RC2042 if the queue is currently open with OOINPX.

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

DistLists (10-digit signed integer)
Distribution list support.

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:

DLSUPP
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.

DLNSUP
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 IADIST selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

HardenGetBackout (10-digit signed integer)
Whether to maintain an accurate backout count.

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 *MDBOC* 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 QABH only if it is essential that the count is accurate.

On OS/400, the message backout count is always hardened, regardless of the setting of this attribute.

The following values are possible:

QABH
Backout count remembered.

Hardening is used to ensure that the backout count for messages on this queue is accurate.

QABNH
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 IAHGB selector with the MQINQ call.

InitiationQName (48-byte character string)
Name of initiation queue.

This is the name of a queue defined on the local queue manager; the queue must be of type QTLOC. 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 CAINIQ selector with the MQINQ call. The length of this attribute is given by LNQN.

MaxMsgLength (10-digit signed integer)
Maximum message length in bytes.

This is the maximum length of the application message data that can exist in each message on the queue. The *MaxMsgLength* local-queue attribute can be set independently of the *MaxMsgLength* queue-manager attribute, and the longest physical message that can be placed on a queue is the lesser of those two values. An attempt to place on the queue a message that is too long fails with reason code:

- RC2030 if the message to too big for the queue
- RC2031 if the message to too big for the queue manager, but not too big for the queue

The value of this attribute is greater than or equal to zero. The upper limit is determined by the environment:

- On OS/400, the maximum message length is 100 MB (104 857 600 bytes).

For more information, see the *BUFLEN* parameter described in "MQPUT – Put message" on page 217.

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

MaxQDepth (10-digit signed integer)
Maximum queue depth.

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 RC2053.

Note: 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 OS/400, 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 IAMDEP selector with the MQINQ call.

MsgDeliverySequence (10-digit signed integer)
Message delivery sequence.

This determines the order in which messages are returned to the application by the MQGET call:

MSPRIO
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).

MSFIFO
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 priority.

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 MSFIFO when the message arrives, the message is placed on the queue as though its priority were *DefPriority*.
- If *MsgDeliverySequence* is MSPRIO when the message arrives, the message is placed on the queue at the place appropriate to priority given by the *MDPRI* field in the message descriptor.

If the value of the *MsgDeliverySequence* attribute is subsequently changed while there are messages on the queue, the order of the messages on the queue is not changed. This does not affect the value of the *MDPRI* field in the MQMD, which retains the value it had when the message was first put.

This means that if the value of the *DefPriority* attribute is changed, messages will not necessarily be delivered in FIFO order, even though the *MsgDeliverySequence* attribute is set to MSFIFO; those that were placed on the queue at the higher priority are delivered first.

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

OpenInputCount (10-digit signed integer)
Number of opens for input.

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.

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.

This attribute does not apply to model queues, but it does apply to the dynamically-defined queues created from the model queue definitions using the MQOPEN call.

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

OpenOutputCount (10-digit signed integer)
Number of opens for output.

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.

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.

This attribute does not apply to model queues, but it does apply to the dynamically-defined queues created from the model queue definitions using the MQOPEN call.

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

ProcessName (48-byte character string)
Process name.

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 CAPRON selector with the MQINQ call. The length of this attribute is given by LNPRON.

QDepthHighEvent (10-digit signed integer)
Controls whether Queue Depth High events are generated.

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. See the description of the Queue Depth High event for more details.

It is one of the following:

EVRDIS
: Event reporting disabled.

EVRENA
: Event reporting enabled.

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

QDepthHighLimit (10-digit signed integer)
: High limit for queue depth.

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 IAQDHL selector with the MQINQ call.

QDepthLowEvent (10-digit signed integer)
: Controls whether Queue Depth Low events are generated.

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. See the description of the Queue Depth Low event for more details.

It is one of the following:

EVRDIS
: Event reporting disabled.

EVRENA
: Event reporting enabled.

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

QDepthLowLimit (10-digit signed integer)
: Low limit for queue depth.

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 IAQDLL selector with the MQINQ call.

QDepthMaxEvent (10-digit signed integer)
Controls whether Queue Full events are generated.

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. See the description of the Queue Full event for more details.

It is one of the following:

EVRDIS
Event reporting disabled.

EVRENA
Event reporting enabled.

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

QServiceInterval (10-digit signed integer)
Target for queue service interval.

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 IAQSI selector with the MQINQ call.

QServiceIntervalEvent (10-digit signed integer)
Controls whether Service Interval High or Service Interval OK events are generated.

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. See the description of the Service Interval High and Service Interval OK events for more details.

It is one of the following:

QSIEHI
Queue Service Interval High events enabled.

- Queue Service Interval High events are **enabled** and
- Queue Service Interval OK events are **disabled**.

QSIEOK
Queue Service Interval OK events enabled.

- Queue Service Interval High events are **disabled** and
- Queue Service Interval OK events are **enabled**.

QSIENO
No queue service interval events enabled.

- Queue Service Interval High events are **disabled** and
- Queue Service Interval OK events are also **disabled**.

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

RetentionInterval (10-digit signed integer)
Retention interval.

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 IARINT selector with the MQINQ call.

Shareability (10-digit signed integer)
Queue shareability.

This indicates whether the queue can be opened for input multiple times concurrently. It is one of the following:

QASHR
Queue is shareable.

Multiple opens with the OOINPS option are allowed.

QANSHR
Queue is not shareable.

An MQOPEN call with the OOINPS option is treated as OOINPX.

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

TriggerControl (10-digit signed integer)
Trigger control.

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:

TCOFF
: Trigger messages not required.

  No trigger messages are to be written for this queue. The value of *TriggerType* is irrelevant in this case.

TCON
: 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 IATRGC selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

TriggerData (64-byte character string)
: Trigger data.

  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 CATRGD selector with the MQINQ call. To change the value of this attribute, use the MQSET call. The length of this attribute is given by LNTRGD.

TriggerDepth (10-digit signed integer)
: Trigger depth.

  This is the number of messages that have to be on the queue before a trigger message is written when *TriggerType* is set to TTDPTH. The value of *TriggerDepth* is one or greater. This attribute is not used otherwise.

  To determine the value of this attribute, use the IATRGD selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

TriggerMsgPriority (10-digit signed integer)
: Threshold message priority for triggers.

  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 "Attributes for the queue-manager" on page 264); a value of zero causes all messages to contribute to the generation of trigger messages.

  To determine the value of this attribute, use the IATRGP selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

TriggerType (10-digit signed integer)
Trigger type.

This controls the conditions under which trigger messages are written as a result of messages arriving on this queue.

It is one of the following:

TTNONE
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 TCOFF.

TTFRST
Trigger message when queue depth goes from 0 to 1.

A trigger message is written whenever the queue changes from empty (no messages on the queue) to not-empty (one or more messages on the queue).

TTEVRY
Trigger message for every message.

A trigger message is written every time a message arrives on the queue.

TTDPTH
Trigger message when depth threshold exceeded.

A trigger message is written when a certain number of messages (*TriggerDepth*) are on the queue. After the trigger message has been written, *TriggerControl* is set to TCOFF to prevent further triggering until it is explicitly turned on again.

To determine the value of this attribute, use the IATRGT selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

Usage (10-digit signed integer)
Queue usage.

This indicates what the queue is used for. It is one of the following:

USNORM
Normal usage.

This is a queue that normal applications use when putting and getting messages; the queue is not a transmission queue.

USTRAN
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 OOOUT 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 "MQXQH –Transmission queue header" on page 159), otherwise errors may occur during the transmission process. Context is not passed or set unless one of the PM* context options is specified.

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

Attributes for local definitions of remote queues

The following table summarizes the attributes that are specific to the local definitions of remote queues. The attributes are described in alphabetic order.

Table 42. Attributes for local definitions of remote queues

Attribute	Description	Page
RemoteQMgrName	Name of remote queue manager	260
RemoteQName	Name of remote queue	261
XmitQName	Transmission queue name	261

A local definition of a remote queue is normally used to refer to a queue that exists on a remote queue manager. It specifies the name of the queue manager at which the queue exists, and optionally the name of the transmission queue to be used to convey messages destined for that queue at that queue manager.

However, the same type of definition can also be used for the following purposes:

- Reply queue aliasing

  The name of the definition is the name of a reply-to queue. For more information, see the *MQSeries Intercommunication* book.

- Queue-manager aliasing

  The name of the definition is actually the alias name of a queue manager, not the name of a queue. For more information, see the *MQSeries Intercommunication* book.

The attributes are described in alphabetic order.

RemoteQMgrName (48-byte character string)
Name of remote queue manager.

The name of the remote queue manager on which the queue *RemoteQName* is defined.

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 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 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 *MDRM*.

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 CARQMN selector with the MQINQ call.

The length of this attribute is given by LNQMN.

RemoteQName (48-byte character string)
Name of remote queue.

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 *MDRQ*.

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 CARQN selector with the MQINQ call.

The length of this attribute is given by LNQN.

XmitQName (48-byte character string)
Transmission queue name.

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, a queue whose name is the same as *RemoteQMgrName* is used instead as the transmission queue.

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 CAXQN selector with the MQINQ call.

The length of this attribute is given by LNQN.

Attributes for alias queues

The following attribute is associated with alias queues:

BaseQName (48-byte character string)
The queue name to which the alias resolves.

This is the name of a queue that is defined to the local queue manager. (For more information on queue names, see the *MQSeries Application Programming Guide*.) The queue is one of the following types:

QTLOC
Local queue.

QTREM
Local definition of a remote queue.

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

The length of this attribute is given by LNQN.

Attributes for process definitions

The following table summarizes the attributes that are specific to process definitions. The attributes are described in alphabetic order.

Table 43. Attributes for process definitions

Attribute	Description	Page
ApplId	Application identifier	262
ApplType	Application type	263
EnvData	Environment data	263
ProcessDesc	Process description	263
ProcessName	Process name	264
UserData	User data	264

The attributes are described in alphabetic order.

ApplId (256-byte character string)
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 interpretation to be placed on this information is determined by the trigger-monitor application.

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 CAAPPI selector with the MQINQ call.

The length of this attribute is given by LNPROA.

ApplType (10-digit signed integer)
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 ATUFST through ATULST:

ATCICS
: CICS transaction.

AT400
: OS/400 application.

ATUFST
: Lowest value for user-defined application type.

ATULST
: Highest value for user-defined application type.

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

EnvData (128-byte character string)
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 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 CAENVD selector with the MQINQ call.

The length of this attribute is given by LNPROE.

ProcessDesc (64-byte character string)
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 CAPROD selector with the MQINQ call.

The length of this attribute is given by LNPROD.

ProcessName (48-byte character string)
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 CAPRON selector with the MQINQ call.

The length of this attribute is given by LNPRON.

UserData (128-byte character string)
User data.

This is a character string that contains user information pertaining to the application to be started.

This information is for use by the 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 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 CAUSRD selector with the MQINQ call.

The length of this attribute is given by LNPROU.

Attributes for the queue-manager

The following table summarizes the attributes that are specific to the queue manager. The attributes are described in alphabetic order.

Table 44. Attributes for the queue manager

Attribute	Description	Page
AuthorityEvent	Controls whether authorization (Not Authorized) events are generated	266
ChannelAutoDef	Controls whether automatic channel definition is permitted	266
ChannelAutoDefEvent	Controls whether channel automatic-definition events are generated	266
ChannelAutoDefExit	Name of user exit for automatic channel definition	266
CodedCharSetId	Coded character set identifier	267
CommandInputQName	Command input queue name	267
CommandLevel	Command level	267
DeadLetterQName	Name of dead-letter queue	268
DefXmitQName	Default transmission queue name	269
DistLists	Distribution list support	269
InhibitEvent	Controls whether inhibit (Inhibit Get and Inhibit Put) events are generated	269
LocalEvent	Controls whether local error events are generated	269
MaxHandles	Maximum number of handles	269
MaxMsgLength	Maximum message length in bytes	270
MaxPriority	Maximum priority	270
MaxUncommittedMsgs	Maximum number of uncommitted messages within a unit of work	270
PerformanceEvent	Controls whether performance-related events are generated	271
Platform	Platform on which the queue manager is running	271
QMgrDesc	Queue manager description	271
QMgrName	Queue manager name	271
RemoteEvent	Controls whether remote error events are generated	272
StartStopEvent	Controls whether start and stop events are generated	272
SyncPoint	Syncpoint availability	272
TriggerInterval	Trigger-message interval	272

Some of these attributes are fixed for particular implementations, others can be changed with the ALTER QMGR command. All can be inquired by opening a special OTQM object, and using the MQINQ call with the handle returned. They can also all be displayed with the DISPLAY QMGR command.

The attributes are described in alphabetic order.

AuthorityEvent (10-digit signed integer)
Controls whether authorization (Not Authorized) events are generated.

It is one of the following:

EVRDIS
Event reporting disabled.

EVRENA
Event reporting enabled.

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

ChannelAutoDef (10-digit signed integer)
Controls whether automatic channel definition is permitted.

This attribute controls the automatic definition of channels of type CTRCVR and CTSVCN. It is one of the following:

CHADDI
Channel auto-definition disabled.

CHADEN
Channel auto-definition enabled.

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

ChannelAutoDefEvent (10-digit signed integer)
Controls whether channel automatic-definition events are generated.

It is one of the following:

EVRDIS
Event reporting disabled.

EVRENA
Event reporting enabled.

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

ChannelAutoDefExit (n-byte character string)
Name of user exit for automatic channel definition.

If this name is nonblank, and *ChannelAutoDef* has the value CHADEN, the exit is called each time that the queue manager is about to create a channel definition. 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 CACADX selector with the MQINQ call.

CodedCharSetId (10-digit signed integer)
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, and queue creation date and time. It must be the identifier of a single-byte character set (SBCS). It does not apply to application data carried in the message.

On OS/400, the value is that which is set in the environment when the queue manager is first created.

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

CommandInputQName (48-byte character string)
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 applications can send commands, if authorized to do so.

On OS/400, 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 CMESC. Refer to the*MQSeries Programmable System Management* book for details of PCF commands.

To determine the value of this attribute, use the CACMDQ selector with the MQINQ call. The length of this attribute is given by LNQN.

CommandLevel (10-digit signed integer)
Command Level.

This indicates the level of system control commands supported by the queue manager. The value is one of the following:

CMLVL1
Level 1 of system control commands.

This value is returned by the following:

- MQSeries for OS/400:
  - Version 2 Release 3
  - Version 3 Release 1
  - Version 3 Release 6

CML320
MQSeries for OS/400 Version 3 Release 2, and Version 3 Release 7.

CML420
MQSeries for AS/400 Version 4 Release 2.

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 IACMDL selector with the MQINQ call.

DeadLetterQName (48-byte character string)

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 *MDRQ* 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 *MDEXP* field described in "MQMD – Message descriptor" on page 59) are **not** transferred to this queue when they are discarded. However, an expiration report message (ROEXP) is still generated and sent to the *MDRQ* 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 "MQDLH – Dead-letter (undelivered-message) header" on page 19 for more details of this structure.

This queue must be a local queue, with a *Usage* attribute of USNORM.

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 CADLQ selector with the MQINQ call. The length of this attribute is given by LNQN.

DefXmitQName (48-byte character string)
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 CADXQN selector with the MQINQ call. The length of this attribute is given by LNQN.

DistLists (10-digit signed integer)
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:

DLSUPP
Distribution lists supported.

DLNSUP
Distribution lists not supported.

To determine the value of this attribute, use the IADIST selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

InhibitEvent (10-digit signed integer)
Controls whether inhibit (Inhibit Get and Inhibit Put) events are generated.

It is one of the following:

EVRDIS
Event reporting disabled.

EVRENA
Event reporting enabled.

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

LocalEvent (10-digit signed integer)
Controls whether local error events are generated.

It is one of the following:

EVRDIS
Event reporting disabled.

EVRENA
Event reporting enabled.

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

MaxHandles (10-digit signed integer)
Maximum number of handles.

This is the maximum number of open handles that any one task can have at the same time.

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

- On OS/400, the default value is 256.

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

MaxMsgLength (10-digit signed integer)
Maximum message length in bytes.

This is the maximum length of the application message data that can be handled by the queue manager. The *MaxMsgLength* queue-manager attribute can be set independently of the *MaxMsgLength* local-queue attribute, and the longest physical message that can be placed on a queue is the lesser of those two values.

The lower limit for this attribute is 32 KB (32 768 bytes). The upper limit is determined by the environment:

- On OS/400, the maximum message length is 100 MB (104 857 600 bytes).

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

MaxPriority (10-digit signed integer)
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 IAMPRI selector with the MQINQ call.

MaxUncommittedMsgs (10-digit signed integer)
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 PMSYP option
- Messages retrieved by the application with the GMSYP option
- Trigger messages and COA report messages generated by the queue manager for messages put with the PMSYP option
- COD report messages generated by the queue manager for messages retrieved with the GMSYP 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 GMSYP)
- Event messages generated by the queue manager (even if the call causing the event message specified PMSYP or GMSYP)

Note: 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.

The lower limit for this attribute is 1; the upper limit is 999 999 999.

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

PerformanceEvent (10-digit signed integer)
Controls whether performance-related events are generated.

It is one of the following:

EVRDIS
Event reporting disabled.

EVRENA
Event reporting enabled.

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

Platform (10-digit signed integer)
Platform on which the queue manager is running.

This indicates the architecture of the platform on which the queue manager is running. The value is:

PL400
OS/400.

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

QMgrDesc (64-byte character string)
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/400, the default value is all blanks.

To determine the value of this attribute, use the CAQMD selector with the MQINQ call. The length of this attribute is given by LNQMD.

QMgrName (48-byte character string)
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 *MDMID* field described in "MQMD – Message descriptor" on page 59). 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.

To determine the value of this attribute, use the CAQMN selector with the MQINQ call. The length of this attribute is given by LNQMN.

RemoteEvent (10-digit signed integer)
Controls whether remote error events are generated.

It is one of the following:

EVRDIS
Event reporting disabled.

EVRENA
Event reporting enabled.

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

StartStopEvent (10-digit signed integer)
Controls whether start and stop events are generated.

It is one of the following:

EVRDIS
Event reporting disabled.

EVRENA
Event reporting enabled.

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

SyncPoint (10-digit signed integer)
Syncpoint availability.

This indicates whether the local queue manager supports units of work and syncpointing with the MQGET, MQPUT, and MQPUT1 calls.

SPAVL
Units of work and syncpointing available.

SPNAVL
Units of work and syncpointing not available.

On OS/400, this value is never returned.

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

TriggerInterval (10-digit signed integer)
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 TTFRST. 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 TTFRST 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 IATRGI selector with the MQINQ call.

Chapter 5. Return codes

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 code

The completion code parameter (*CMPCOD*) 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:

CCOK
: Successful completion.

  The call completed fully; all output parameters have been set. The *REASON* parameter always has the value RCNONE in this case.

CCWARN
: Warning (partial completion).

  The call completed partially. Some output parameters may have been set in addition to the *CMPCOD* and *REASON* output parameters. The *REASON* parameter gives additional information about the partial completion.

CCFAIL
: 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 *CMPCOD* 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 application's authority may have been revoked. The *REASON* parameter gives additional information about the error.

Reason code

The reason code parameter (*REASON*) is a qualification to the completion code parameter (*CMPCOD*).

If there is no special reason to report, RCNONE is returned. A successful call returns CCOK and RCNONE.

If the completion code is either CCWARN or CCFAIL, 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.

© Copyright IBM Corp. 1994, 1998

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 *DLREA* field of the MQDLH structure (for messages on the dead-letter queue)
- The *MDFB* field of the MQMD structure (message descriptor)

The following is a list of reason codes, in alphabetic order, with more detail than is given in the call descriptions.

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

  The call completed normally. The completion code (*CMPCOD*) is CCOK.

  Corrective action: None.

RC2001
: (2001, X'7D1') Alias base queue not a valid type.

  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, or local definition of a remote queue.

  Corrective action: Correct the queue definitions.

RC2002
: (2002, X'7D2') Application already connected.

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

  Corrective action: None. The *HCONN* parameter returned has the same value as was returned for the previous MQCONN call.

  Note: An MQCONN 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 (or portion thereof) 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 call to be disconnected as well.

RC2004
: (2004, X'7D4') Buffer parameter not valid.

  BUFFER is not valid. The parameter pointer is not valid, or points to read-only storage for MQGET calls, or to storage that cannot be accessed for the entire length specified by *BUFLEN*. (It is not always possible to detect parameter pointers that are not valid; if it is not detected, unpredictable results occur.)

  Corrective action: Correct the parameter.

RC2005
: (2005, X'7D5') Buffer length parameter not valid.

  BUFLEN or the parameter pointer is not valid. (It is not always possible to detect an invalid parameter pointer; if not detected, unpredictable results occur.)

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

Corrective action: Specify a nonnegative value.

RC2006
(2006, X'7D6') Length of character attributes not valid.

CALEN 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 an invalid parameter pointer; if not detected, unpredictable results occur.)

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

RC2007
(2007, X'7D7') Character attributes string not valid.

CHRATR 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 *CALEN*. (It is not always possible to detect parameter pointers that are not valid; if it is not detected, unpredictable results occur.)

Corrective action: Correct the parameter.

RC2008
(2008, X'7D8') Not enough space allowed for character attributes.

For MQINQ calls, *CALEN* is not large enough to contain all of the character attributes for which CA* selectors are specified in the *SELS* parameter.

The call still completes, with the *CHRATR* 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.

Corrective action: Specify a large enough value, unless only a subset of the values is needed.

RC2009
(2009, X'7D9') Connection to queue manager lost.

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 GMWT option, the wait has been canceled.

If this reason occurs with MQCONN, the queue manager may have been stopped and restarted, and now be available again. All previous handles are now invalid, but the application can attempt to reestablish connection by issuing MQCONN again.

Note that for MQ client applications it is possible that the call did complete successfully, even though this reason code is returned with a *CMPCOD* of CCFAIL.

Corrective action: Applications can attempt to reestablish connection by issuing the MQCONN call. It may be necessary to poll until a successful response is received.

Applications should ensure that any uncommitted updates are backed out. Any unit of work that is coordinated by the queue manager is backed out automatically.

RC2010
: (2010, X'7DA') Data length parameter not valid.

DATLEN is not valid. The parameter pointer is not valid, or points to read-only storage. (It is not always possible to detect an invalid parameter pointer; if not detected, unpredictable results occur.)

This reason can also be returned to an MQ client program on the MQGET, MQPUT, or MQPUT1 call, if the application message data is longer than the negotiated maximum message size for the channel.

Corrective action: 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.

RC2011
: (2011, X'7DB') Name of dynamic queue not valid.

On the MQOPEN call, a model queue is specified in the *ODON* field of the *OBJDSC* parameter, but the *ODDN* field is not valid, for one of the following reasons:

- 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 which are not null and not blank.

Corrective action: Specify a valid name.

RC2013
: (2013, X'7DD') Expiry time not valid.

On an MQPUT or MQPUT1 call, the value specified for the *MDEXP* field in the message descriptor MQMD is not valid.

Corrective action: Specify a value which is greater than zero, or the special value EIULIM.

RC2014
: (2014, X'7DE') Feedback code not valid.

On an MQPUT or MQPUT1 call, the value specified for the *MDFB* field in the message descriptor MQMD is not valid. The value is outside both the range defined for system feedback codes and that defined for application feedback codes.

Corrective action: Specify a value in the range FBSFST through FBSLST, or FBAFST through FBALST.

RC2016
: (2016, X'7E0') Gets inhibited for the queue.

MQGET calls are currently inhibited for the queue (see the *InhibitGet* queue attribute described in “Attributes for all queues” on page 243), or for the

queue to which this queue resolves (see "Attributes for alias queues" on page 262).

Corrective action: If the system design allows get requests to be inhibited for short periods, retry the operation later.

RC2017
(2017, X'7E1') No more handles available.

An MQOPEN or MQPUT1 call was issued, but the maximum number of open handles allowed for the current task has already been reached.

Corrective action: Check whether the application is issuing MQOPEN calls without corresponding MQCLOSE calls. If it is not, reduce the complexity of the application. The maximum number of open handles that a task can have is given by the *MaxHandles* queue manager attribute (see "Attributes for the queue-manager" on page 264).

RC2018
(2018, X'7E2') Connection handle not valid.

The connection handle *HCONN* is not valid. This reason also occurs if the parameter pointer is not valid, or (for the MQCONN call) points to read-only storage. (It is not always possible to detect an invalid parameter pointer; if not detected, unpredictable results occur.)

Corrective action: Ensure that a successful MQCONN 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 "MQCONN – Connect queue manager" on page 175).

RC2019
(2019, X'7E3') Object handle not valid.

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 an invalid parameter pointer; if not detected, unpredictable results occur.)

Corrective action: 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 "MQOPEN – Open object" on page 204).

RC2020
(2020, X'7E4') Value for inhibit-get or inhibit-put queue attribute not valid.

On an MQSET call, the value specified for either the IAIGET attribute or the IAIPUT attribute is not valid.

Corrective action: Specify a valid value. See the *InhibitGet* or *InhibitPut* attribute described in "Attributes for all queues" on page 243.

RC2021
(2021, X'7E5') Count of integer attributes not valid.

On an MQINQ or MQSET call, the *IACNT* parameter is negative (MQINQ or MQSET), or smaller than the number of integer attribute selectors (IA*) specified in the *SELS* parameter (MQSET only). This reason also occurs if the

parameter pointer is not valid. (It is not always possible to detect an invalid parameter pointer; if not detected, unpredictable results occur.)

Corrective action: Specify a value large enough for all selected integer attributes.

RC2022
(2022, X'7E6') Not enough space allowed for integer attributes.

On an MQINQ call, the *IACNT* parameter is smaller than the number of integer attribute selectors (IA*) specified in the *SELS* parameter.

The call completes with CCWARN, with the *INTATR* array filled in with as many integer attributes as there is room for.

Corrective action: Specify a large enough value, unless only a subset of the values is needed.

RC2023
(2023, X'7E7') Integer attributes array not valid.

On an MQINQ or MQSET call, the *INTATR* 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 *IACNT* parameter (MQINQ only). (It is not always possible to detect an invalid parameter pointer; if not detected, unpredictable results occur.)

Corrective action: Correct the parameter.

RC2024
(2024, X'7E8') No more messages can be handled within current unit of work.

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 PMSYP option
- Messages retrieved by the application with the GMSYP option
- Trigger messages and COA report messages generated by the queue manager for messages put with the PMSYP option
- COD report messages generated by the queue manager for messages retrieved with the GMSYP option

Corrective action: 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/400, the limit for the maximum number of uncommitted messages can be changed by using the CHGMQM command.

RC2026
(2026, X'7EA') Message descriptor not valid.

MQMD structure is not valid. Either the *MDSID* mnemonic eye-catcher is not valid, or the *MDVER* is not recognized.

This reason also occurs if:

- The parameter pointer is not valid. (It is not always possible to detect an invalid parameter pointer; 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.

Corrective action: Correct the definition of the message descriptor. Ensure that required input fields are correctly set.

RC2027
: (2027, X'7EB') Missing reply-to queue.

On an MQPUT or MQPUT1 call, the *MDRQ* field in the message descriptor MQMD is blank, but one or both of the following is true:

- A reply was requested (that is, MTRQST was specified in the *MDMT* field of the message descriptor).
- A report message was requested in the *MDREP* field of the message descriptor.

Corrective action: Specify the name of the queue to which the reply message or report message is to be sent.

RC2029
: (2029, X'7ED') Message type in message descriptor not valid.

On an MQPUT or MQPUT1 call, the value specified for the *MDMT* field in the message descriptor (MQMD) is not valid.

Corrective action: Specify a valid value. See the *MDMT* field described in "MQMD – Message descriptor" on page 59 for details.

RC2030
: (2030, X'7EE') Message length greater than maximum for queue.

On an MQPUT or MQPUT1 call, an attempt was made to put onto a queue a message that is bigger than the maximum allowed for that queue (see the *MaxMsgLength* local-queue attribute described in in "Attributes for local queues" on page 246).

This reason code can also occur in the *MDFB* 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.

Corrective action: Check whether the *BUFLEN* parameter was specified correctly. If so, either break the message into several smaller messages, or increase *MaxMsgLength* for the queue.

RC2031
: (2031, X'7EF') Message length greater than maximum for queue manager.

On an MQPUT or MQPUT1 call, an attempt was made to put onto a queue a message that is bigger than the maximum allowed by the queue manager (see the *MaxMsgLength* queue-manager attribute described in in "Attributes for the queue-manager" on page 264).

This reason code can also occur in the *MDFB* 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.

Corrective action: Check whether the *BUFLEN* parameter was specified correctly. If it was, break the message into several smaller messages. Check the channel definitions.

RC2033
: (2033, X'7F1') No message available.

An MQGET call was issued, but there is no message on the queue satisfying the selection criteria specified in MQMD (the *MDMID* and *MDCID* fields), and in MQGMO (the *GMOPT* and *GMMO* fields). Either the GMWT option was not specified, or the time interval specified by the *GMWI* 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.

Corrective action: If this is an expected condition, no corrective action is required.

If this is an unexpected condition, check whether the message was put on the queue successfully, and whether 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:

GMLOGO
GMAMSA
GMASGA
GMCMPM
MOMSGI
MOCORI
MOGRPI
MOSEQN
MOOFFS
MDMID field
MDCID field

Consider waiting longer for the message.

RC2034
: (2034, X'7F2') Browse cursor not positioned on message.

An MQGET call was issued with either the GMMUC or the GMBRWC 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), or
- 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.

Corrective action: 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 GMLK option with the preceding browse MQGET call.

RC2035
(2035, X'7F3') Not authorized for access.

The user is not authorized to perform the operation attempted:

- On an MQCONN call, the user is not authorized to connect to the queue manager.
- On an MQOPEN or MQPUT1 call, the user is not authorized to open the object for the option(s) specified.
- 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 *MDFB* 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.

Corrective action: Ensure that the correct queue manager or object was specified, and that appropriate authority exists.

RC2036
(2036, X'7F4') Queue not open for browse.

An MQGET call was issued with one of the following options:

GMBRWF
GMBRWN
GMBRWC
GMMUC

but the queue had not been opened for browse.

Corrective action: Specify OOBRW when the queue is opened.

RC2037
(2037, X'7F5') Queue not open for input.

An MQGET call was issued to retrieve a message from a queue, but the queue had not been opened for input.

Corrective action: Specify one of the following when the queue is opened:

OOINPS
OOINPX
OOINPQ

RC2038
(2038, X'7F6') Queue not open for inquire.

An MQINQ call was issued to inquire object attributes, but the object had not been opened for inquire.

Corrective action: Specify OOINQ when the object is opened.

RC2039
(2039, X'7F7') Queue not open for output.

An MQPUT call was issued to put a message on a queue, but the queue had not been opened for output.

Corrective action: Specify OOOUT when the queue is opened.

RC2040
(2040, X'7F8') Queue not open for set.

An MQSET call was issued to set queue attributes, but the queue had not been opened for set.

Corrective action: Specify OOSET when the object is opened.

RC2041
(2041, X'7F9') Object definition changed since opened.

Since the *HOBJ* handle used on this call was returned by the MQOPEN call, object definitions that affect this object have been changed. See "MQOPEN – Open object" on page 204 for more information.

This reason does not occur if the object handle is specified in the *PMCT* field of the *PMO* parameter on the MQPUT or MQPUT1 call.

Corrective action: 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 find out what has changed.

RC2042
(2042, X'7FA') Object already open with conflicting options.

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 *OPTS* 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).

Note: MCAs for receiver channels may keep the destination queues open even when messages are not being transmitted; this results in the queues appearing to be "in use."

Corrective action: System design should specify whether an application is to wait and retry, or take other action.

RC2043
(2043, X'7FB') Object type not valid.

On the MQOPEN or MQPUT1 call, the *ODOT* field in the object descriptor MQOD specifies a value which is not valid. For the MQPUT1 call, the object type must be OTQ.

Corrective action: Specify a valid object type.

RC2044
(2044, X'7FC') Object descriptor structure not valid.

On the MQOPEN or MQPUT1 call, the object descriptor MQOD is not valid. Either the *ODSID* mnemonic eye-catcher is not valid, or the *ODVER* is not recognized.

This reason also occurs if:

- The parameter pointer is not valid. (It is not always possible to detect an invalid parameter pointer; 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.

Corrective action: Correct the definition of the object descriptor. Ensure that required input fields are set correctly.

RC2045
(2045, X'7FD') Option not valid for object type.

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 both an option that is inappropriate to the object type (for example, OOOUT for an OTPRO object), and one that is unsupported for the queue type (for example, OOINQ for a remote queue that has no local definition).

This reason also occurs on the MQOPEN call if one of the following is true:

- the queue name is resolved through a cell directory, or
- *ODMN* 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,

and the options include one of the following:

OOINPQ
OOINPS
OOINPX
OOBRW
OOINQ
OOSET

For the MQCLOSE call, this reason code occurs when the CODEL or COPURG option was specified, but the queue is not a dynamic queue.

Corrective action: Specify the correct option; see Table 39 on page 209 for open options, and Table 38 on page 171 for close options. For the MQCLOSE call, either correct the option or change the definition type of the model queue that was used to create the queue.

RC2046
(2046, X'7FE') Options not valid or not consistent.

The *OPTS* parameter or field contains options which are not valid, or a combination of options which is not valid.

- For the MQOPEN, MQCLOSE, and MQXCNVC calls, *GMOPT* 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 an invalid parameter pointer; if not detected, unpredictable results occur.)

- For the MQGET, MQPUT, and MQPUT1 calls, *GMOPT* is a field in the relevant options structure (MQGMO or MQPMO).

Corrective action: Specify valid options. Check the description of the *OPTS* 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.

RC2047

(2047, X'7FF') Persistence not valid.

On an MQPUT or MQPUT1 call, the value specified for the *MDPER* field in the message descriptor MQMD is not valid.

Corrective action: Specify one of the following values:

PEPER
PENPER
PEQDEF

RC2048

(2048, X'800') Message on a temporary dynamic queue cannot be persistent.

On an MQPUT or MQPUT1 call, the value specified for the *MDPER* field in the message descriptor MQMD specifies PEPER, but the queue on which the message is being placed is a temporary dynamic queue. Persistent messages cannot be put on temporary queues.

This reason code can also occur in the *MDFB* 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.

Corrective action: Specify PENPER if the message is to be placed on a temporary dynamic queue. If persistence is required, use a permanent dynamic queue, or a predefined queue.

Be aware that server applications are recommended to send reply messages (message type MTRPLY) with the same persistence as the original request message (message type MTRQST). If the request message is persistent, the reply queue specified in the *MDRQ* 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.

RC2049

(2049, X'801') Message Priority exceeds maximum value supported.

On an MQPUT or MQPUT1 call, the value of the *MDPRI* field in the message descriptor MQMD exceeds the maximum priority supported by the local queue manager (see the *MaxPriority* queue-manager attribute described in "Attributes for the queue-manager" on page 264). The message is accepted by the queue manager, but is placed on the queue at the queue manager's maximum priority. The *MDPRI* field in the message descriptor retains the value specified by the application that put the message.

Corrective action: None required, unless this reason code was not expected by the application that put the message.

RC2050

(2050, X'802') Message priority not valid.

On an MQPUT or MQPUT1 call, the value of the *MDPRI* field in the message descriptor MQMD is not valid.

Corrective action: Specify a value which is zero or greater, or the special value PRQDEF.

RC2051

(2051, X'803') Put calls inhibited for the queue.

MQPUT and MQPUT1 calls are currently inhibited for the queue (see the *InhibitPut* queue attribute described in “Attributes for all queues” on page 243), or for the queue to which this queue resolves (see “Attributes for alias queues” on page 262).

This reason code can also occur in the *MDFB* 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.

Corrective action: If the system design allows put requests to be inhibited for short periods, retry the operation later.

RC2052

(2052, X'804') Queue has been deleted.

An *HOBJ* queue handle specified on a call refers to a dynamic queue that has been deleted since the queue was opened. (See “MQCLOSE – Close object” on page 169 for information about the deletion of dynamic queues.)

Corrective action: 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.

RC2053

(2053, X'805') Queue already contains maximum number of messages.

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* local-queue attribute described in “Attributes for local queues” on page 246).

This reason code can also occur in the *MDFB* 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.

Corrective action: Retry the operation later. Consider increasing the maximum depth for this queue, or arranging for more instances of the application to service the queue.

RC2055

(2055, X'807') Queue contains one or more messages or uncommitted put or get requests.

An MQCLOSE call was issued for a permanent dynamic queue, with either:

- The CODEL option specified, but there are messages still on the queue, or

- The CODEL or COPURG option 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).

Corrective action: Check why there might be messages on the queue. Be aware that the *CurrentQDepth* local-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 which has not yet been committed. If the messages can be discarded, try using the MQCLOSE call with the COPURG option. Consider retrying the call later.

RC2056
: (2056, X'808') No space available on disk for queue.

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 *MDFB* 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.

Corrective action: Check whether an application is putting messages in an infinite loop. If not, make more disk space available for the queue.

On OS/400, the space available for a queue is limited to 320 MB. If this limit has been reached, consider redesigning the application to reduce the number or size of messages on a single queue, or start more server instances.

RC2057
: (2057, X'809') Queue type not valid.

One of the following occurred:

- On an MQOPEN call, the *ODMN* 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 *ODON* 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 *MDRQ* 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 RC2057 is returned in the *DLREA* field of the MQDLH structure when the reply message or report message is placed on the dead-letter queue.

Corrective action: Specify a valid queue.

RC2058

(2058, X'80A') Queue manager name not valid or not known.

On an MQCONN call, the value specified for the *QMNAME* parameter is not valid. This reason also occurs if the parameter pointer is not valid. (It is not always possible to detect an invalid parameter pointer; if not detected, unpredictable results occur.)

This reason also occurs if an application attempts to connect to a queue manager within a group (see the *QMNAME* parameter of MQCONN), and either:

- Queue-manager groups are not supported (they are only supported for MQ client applications), or
- There is no queue-manager group with the specified name.

Corrective action: Use an all-blank name if possible, or verify that the name used is valid.

RC2059

(2059, X'80B') Queue manager not available for connection.

On an MQCONN call, the queue manager identified by the *QMNAME* parameter is not available for connection at this time.

- On OS/400, this reason can also be returned by the MQOPEN and MQPUT1 calls, when HCDEFH is specified for the *HCONN* parameter.

If the connection is from an MQ client application, this reason code can occur if there is an error with the client-connection or the corresponding server-connection channel definitions.

This reason also occurs if an application attempts to connect to a queue manager within a group (see the *QMNAME* parameter of MQCONN), when none of the queue managers in the group is available for connection at this time.

Corrective action: Ensure that the queue manager has been started. If the connection is from a client application, check the channel definitions.

RC2061

(2061, X'80D') Report options in message descriptor not valid.

An MQPUT or MQPUT1 call was issued, but the *MDREP* field in the message descriptor MQMD contains one or more options which 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 C, "Report options" on page 379 for more details.

This reason code can also occur in the *MDFB* field in the MQMD of a report message, or in the *DLREA* 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.

Corrective action: Do the following:

1. Ensure that the *MDREP* 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 RONONE if no report options are required.

2. Ensure that the report options specified are ones which are documented in this book; see the *MDREP* field described in "MQMD – Message descriptor" on page 59 for valid report options. Remove any report options which are not documented in this book.
3. If multiple report options are being set by adding the individual report options together, ensure that the same report option is not added twice.
4. Check that conflicting report options are not specified. For example, do not add both ROEXC and ROEXCD to the *MDREP* field; only one of these can be specified.

RC2063
: (2063, X'80F') Security error occurred.

An MQOPEN, MQPUT1, or MQCLOSE call was issued, but it failed because a security error occurred.

Corrective Action: Note the error from the security manager, and contact your system programmer or security administrator.

On OS/400, the FFST log will contain the error information.

RC2065
: (2065, X'811') Count of selectors not valid.

On an MQINQ or MQSET call, the *SELCNT* 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 an invalid parameter pointer; if not detected, unpredictable results occur.)

Corrective action: Specify a value in the range 0 through 256.

RC2066
: (2066, X'812') Count of selectors too big.

On an MQINQ or MQSET call, the *SELCNT* parameter specifies a value that is larger than the maximum supported (256).

Corrective action: Reduce the number of selectors specified on the call; the valid range is 0 through 256.

RC2067
: (2067, X'813') Attribute selector not valid.

On an MQINQ or MQSET call, a selector in the *SELS* array is either:

- not valid, or
- not applicable to the type of the object whose attributes are being inquired or set, or
- (MQSET only) not an attribute which can be set.

This reason also occurs if the parameter pointer is not valid. (It is not always possible to detect an invalid parameter pointer; if not detected, unpredictable results occur.)

Corrective action: 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.

RC2068

(2068, X'814') Selector not applicable to queue type.

On the MQINQ call, one or more selectors in the *SELS* array is not applicable to the type of the queue whose attributes are being inquired. The call completes with CCWARN, with the attribute values for the inapplicable selectors set as follows:

- For integer attributes, the corresponding elements of *INTATR* are set to IAVNA.
- For character attributes, the appropriate parts of the *CHRATR* string are set to a character string consisting entirely of asterisks (*).

Corrective action: Verify that the selector specified is the one that was intended.

RC2071

(2071, X'817') Insufficient storage available.

A call cannot complete because sufficient storage is not available to the queue manager.

Corrective action: Ensure that active applications are behaving correctly, for example, that they are not looping. If no problems are found, make more storage available.

RC2072

(2072, X'818') Syncpoint support not available.

GMSYP was specified on an MQGET call, or PMSYP 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 SPNAVL.

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 RC2072 is returned.

- On OS/400, this reason codes means that OS/400 Commitment Control is not started, or is unavailable for use by the queue manager.

Corrective action: Remove the specification of GMSYP or PMSYP, as appropriate.

On OS/400, if Commitment Control has not been started, start it. If this reason code occurs after Commitment Control has been started, contact your systems programmer.

RC2075

(2075, X'81B') Value for trigger-control attribute not valid.

On an MQSET call, the value specified for the IATRGC attribute selector is not valid.

Corrective action: Specify a valid value. See “Attributes for local queues” on page 246.

RC2076
: (2076, X'81C') Value for trigger-depth attribute not valid.

On an MQSET call, the value specified for the IATRGD attribute selector is not valid.

Corrective action: Specify a value which is greater than zero. See "Attributes for local queues" on page 246.

RC2077
: (2077, X'81D') Value for trigger-message-priority attribute not valid.

On an MQSET call, the value specified for the IATRGP attribute selector is not valid.

Corrective action: Specify a value in the range 0 through the value of *MaxPriority* queue-manager attribute. See "Attributes for local queues" on page 246.

RC2078
: (2078, X'81E') Value for trigger-type attribute not valid.

On an MQSET call, the value specified for the IATRGT attribute selector is not valid.

Corrective action: Specify a valid value. See "Attributes for local queues" on page 246.

RC2079
: (2079, X'81F') Truncated message returned (processing completed).

On an MQGET call, the message length was too large to fit into the supplied buffer. The GMATM 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 *DATLEN* 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.

Corrective action: None, because the application expected this situation.

RC2080
: (2080, X'820') Truncated message returned (processing not completed).

On an MQGET call, the message length was too large to fit into the supplied buffer. The GMATM 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 GMBRWF was specified, the browse cursor is positioned logically before the highest-priority message on the queue.

The *DATLEN* 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.

Corrective action: Supply a buffer that is at least as large as *DATLEN*, or specify GMATM if not all of the message data is required.

RC2082
: (2082, X'822') Unknown alias base queue.

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.

Corrective action: Correct the queue definitions.

RC2085
: (2085, X'825') Unknown object name.

On an MQOPEN or MQPUT1 call, the *ODMN* field in the object descriptor MQOD is set to 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

However, the *ODON* field in the object descriptor is not recognized for the specified object type.

See also RC2052.

Corrective action: 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.

RC2086
: (2086, X'826') Unknown object queue manager.

On an MQOPEN or MQPUT1 call, the *ODMN* 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 *ODOT* field in the object descriptor has the value OTQM, and the *ODMN* field is not blank, but the name specified is not the name of the local queue manager.

Corrective Action: Specify a valid queue manager name (or all blanks or an initial null character to refer to the local queue manager). Ensure that the name is padded to the right with blanks or terminated with a null character if necessary.

RC2087
: (2087, X'827') Unknown remote queue manager.

On an MQOPEN or MQPUT1 call, an error occurred with the queue-name resolution, for one of the following reasons:

- *ODMN* is either blank or the name of the local queue manager, and *ODON* is the name of a local definition of a remote queue, which has a blank *XmitQName*. However, there is no (transmission) queue defined with the name of *RemoteQMgrName*, and the *DefXmitQName* queue-manager attribute is blank.
- *ODMN* is the name of a queue-manager alias definition (held as the local definition of a remote queue), which has a blank *XmitQName*. However, there is no (transmission) queue defined with the name of *RemoteQMgrName*, and the *DefXmitQName* queue-manager attribute is blank.
- *ODMN* specified is not:
  - Blank

- The name of the local queue manager
- The name of a local queue
- The name of a queue-manager alias definition (that is, a local definition of a remote queue with a blank *RemoteQName*)

and the *DefXmitQName* queue-manager attribute is blank.

- *ODMN* is blank or is the name of the local queue manager, and *ODON* is the name of a local definition of a remote queue (or an alias to one), for which *RemoteQMgrName* is either blank or is the name of the local queue manager. Note that this error occurs even if the *XmitQName* is not blank.
- *ODMN* is the name of a local definition of a remote queue. In this context, this should be a queue-manager alias definition, but the *RemoteQName* in the definition is not blank.
- *ODMN* 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. Also, the *DefXmitQName* queue-manager attribute is blank.

Corrective action: Check the values specified for *ODMN* and *ODON*. If these are correct, check the queue definitions.

RC2090

(2090, X'82A') Wait interval in MQGMO not valid.

On the MQGET call, the value specified for the *GMWI* field in the *GMO* parameter is not valid.

Corrective action: Specify a value greater than or equal to zero, or the special value WIULIM if an indefinite wait is required.

RC2091

(2091, X'82B') Transmission queue not local.

On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager. The *ODON* or *ODMN* 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.

Corrective action: Check the values specified for *ODON* and *ODMN*. If these are correct, check the queue definitions. For more information on transmission queues, see the *MQSeries Application Programming Guide*.

RC2092

(2092, X'82C') Transmission queue with wrong usage.

On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager, but one of the following occurred:

- *ODMN* specifies the name of a local queue, but it does not have a *Usage* attribute of USTRAN.

- The *ODON* or *ODMN* 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 USTRAN
  - *XmitQName* is blank, but *RemoteQMgrName* specifies a queue that does not have a *Usage* attribute of USTRAN
- 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 USTRAN.

Corrective action: Check the values specified for *ODON* and *ODMN*. If these are correct, check the queue definitions. For more information on transmission queues, see the *MQSeries Application Programming Guide*.

RC2093
(2093, X'82D') Queue not open for pass all context.

An MQPUT call was issued with the PMPASA option specified in the *PMO* parameter, but the queue had not been opened with the OOPASA option.

Corrective action: Specify OOPASA (or another option that implies it) when the queue is opened.

RC2094
(2094, X'82E') Queue not open for pass identity context.

An MQPUT call was issued with the PMPASI option specified in the *PMO* parameter, but the queue had not been opened with the OOPASI option.

Corrective action: Specify OOPASI (or another option that implies it) when the queue is opened.

RC2095
(2095, X'82F') Queue not open for set all context.

An MQPUT call was issued with the PMSETA option specified in the *PMO* parameter, but the queue had not been opened with the OOSETA option.

Corrective action: Specify OOSETA when the queue is opened.

RC2096
(2096, X'830') Queue not open for set identity context.

An MQPUT call was issued with the PMSETI option specified in the *PMO* parameter, but the queue had not been opened with the OOSETI option.

Corrective action: Specify OOSETI (or another option that implies it) when the queue is opened.

RC2097
(2097, X'831') Queue handle referred to does not save context.

On an MQPUT or MQPUT1 call, PMPASI or PMPASA was specified, but the handle specified in the *PMCT* field of the *PMO* parameter is either not a valid queue handle, or it is a valid queue handle but the queue was not opened with OOSAVA.

Corrective action: Specify OOSAVA when the queue referred to is opened.

RC2098
(2098, X'832') Context not available for queue handle referred to.

On an MQPUT or MQPUT1 call, PMPASI or PMPASA was specified, but the queue handle specified in the *PMCT* field of the *PMO* 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.

Corrective action: Ensure that a successful nonbrowse get call has been issued with the queue handle referred to.

RC2100
(2100, X'834') Object already exists.

An MQOPEN call was issued to create a dynamic queue, but a queue with the same name as the dynamic queue already exists.

Corrective action: 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.

RC2101
(2101, X'835') Object damaged.

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.

Corrective action: It may be necessary to stop and restart the queue manager, or to restore the queue-manager data from back-up storage.

Consult the FFST record to obtain more detail about the problem.

RC2102
(2102, X'836') Insufficient system resources available.

There are insufficient system resources to complete the call successfully.

Corrective action: Run the application when the machine is less heavily loaded.

Consult the FFST record to obtain more detail about the problem.

RC2104
(2104, X'838') Report option(s) in message descriptor not recognized.

An MQPUT or MQPUT1 call was issued, but the *MDREP* field in the message descriptor MQMD contains one or more options which 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 C, “Report options” on page 379 for more details.

Corrective action: If this reason code is expected, no corrective action is required.

If this reason code is not expected, do the following:

1. Ensure that the *MDREP* 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.
2. Ensure that the report options specified are ones which are documented in this book; see the *MDREP* field described in “MQMD – Message descriptor” on page 59 for valid report options. Remove any report options which are not documented in this book.
3. If multiple report options are being set by adding the individual report options together, ensure that the same report option is not added twice.
4. Check that conflicting report options are not specified. For example, do not add both ROEXC and ROEXCD to the *MDREP* field; only one of these can be specified.

RC2110

(2110, X'83E') Message format not valid.

On an MQGET call with the GMCONV option included in the *GMO* parameter, one or both of the *MDCSI* and *MDENC* fields in the message differs from the corresponding field in the *MSGDSC* parameter, but the message cannot be converted successfully due to an error associated with the message format. Possible errors include:

- A user-written exit with the name specified by the *MDFMT* field in the message cannot be found.
- The format name in the message is FMNONE.
- 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 *MDCSI* and *MDENC* fields in the *MSGDSC* parameter are set to those of the message returned, and the call completes with CCWARN.

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 FMDLH), 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.

Corrective action: 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.

RC2111
(2111, X'83F') Source coded character set identifier not valid.

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 GMCONV option is included in the *GMO* parameter; the coded character-set identifier in error is the *MDCSI* field in the message being retrieved. In this case, the message data is returned unconverted, the values of the *MDCSI* and *MDENC* fields in the *MSGDSC* parameter are set to those of the message returned, and the call completes with CCWARN.

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 FMDLH), 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; the coded character-set identifier in error is the *SRCCSI* parameter. Either the *SRCCSI* parameter specifies a value which is not valid or not supported, or the *SRCCSI* parameter pointer is not valid. (It is not always possible to detect an invalid parameter pointer; if not detected, unpredictable results occur.)

Corrective action: Check the character-set identifier that was specified when the message was put, or that was specified for the *SRCCSI* 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.

RC2112
(2112, X'840') Source integer encoding not recognized.

On an MQGET call, with the GMCONV option included in the *GMO* parameter, the *MDENC* value in the message being retrieved specifies an integer encoding that is not recognized. The message data is returned unconverted, the values of the *MDCSI* and *MDENC* fields in the *MSGDSC* parameter are set to those of the message returned, and the call completes with CCWARN.

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 FMDLH), 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 can also occur on the MQXCNVC call, when the *OPTS* parameter contains an unsupported DCCS* value, or when DCCSUN is specified for a UCS2 code page.

Corrective action: 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.

RC2113
: (2113, X'841') Packed-decimal encoding in message not recognized.

On an MQGET call with the GMCONV option included in the *GMO* parameter, the *MDENC* value in the message being retrieved specifies a decimal encoding that is not recognized. The message data is returned unconverted, the values of the *MDCSI* and *MDENC* fields in the *MSGDSC* parameter are set to those of the message returned, and the call completes with CCWARN.

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 FMDLH), 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.

Corrective action: 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.

RC2114
: (2114, X'842') Floating-point encoding in message not recognized.

On an MQGET call, with the GMCONV option included in the *GMO* parameter, the *MDENC* 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 *MDCSI* and *MDENC* fields in the *MSGDSC* parameter are set to those of the message returned, and the call completes with CCWARN.

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 FMDLH), 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.

Corrective action: 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.

RC2115
: (2115, X'843') Target coded character set identifier not valid.

The coded character-set identifier to which character data which is to be converted is not valid or not supported.

This can occur on the MQGET call when the GMCONV option is included in the *GMO* parameter; the coded character-set identifier in error is the *MDCSI* field in the *MSGDSC* parameter. In this case, the message data is returned unconverted, the values of the *MDCSI* and *MDENC* fields in the *MSGDSC* parameter are set to those of the message returned, and the call completes with CCWARN.

This reason can also occur on the MQXCNVC call; the coded character-set identifier in error is the *TGTCSI* parameter. Either the *TGTCSI* parameter specifies a value which is not valid or not supported, or the *TGTCSI* parameter pointer is not valid. (It is not always possible to detect an invalid parameter pointer; if not detected, unpredictable results occur.)

Corrective action: Check the character-set identifier that was specified for the *MDCSI* field in the *MSGDSC* parameter on the MQGET call, or that was specified for the *SRCCSI* 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.

RC2116
: (2116, X'844') Target integer encoding not recognized.

On an MQGET call with the GMCONV option included in the *GMO* parameter, the *MDENC* value in the *MSGDSC* parameter specifies an integer encoding that is not recognized. The message data is returned unconverted, the values of the *MDCSI* and *MDENC* fields in the *MSGDSC* parameter are set to those of the message being retrieved, and the call completes with CCWARN.

This reason code can also occur on the MQXCNVC call, when the *OPTS* parameter contains an unsupported DCCT* value, or when DCCTUN is specified for a UCS2 code page.

Corrective action: 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.

RC2117
: (2117, X'845') Packed-decimal encoding specified by receiver not recognized.

On an MQGET call with the GMCONV option included in the *GMO* parameter, the *MDENC* value in the *MSGDSC* parameter specifies a decimal encoding that is not recognized. The message data is returned unconverted, the values of the *MDCSI* and *MDENC* fields in the *MSGDSC* parameter are set to those of the message returned, and the call completes with CCWARN.

Corrective action: 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.

RC2118
: (2118, X'846') Floating-point encoding specified by receiver not recognized.

On an MQGET call with the GMCONV option included in the *GMO* parameter, the *MDENC* value in the *MSGDSC* parameter specifies a floating-point encoding that is not recognized. The message data is returned unconverted, the values of the *MDCSI* and *MDENC* fields in the *MSGDSC* parameter are set to those of the message returned, and the call completes with CCWARN.

Corrective action: 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.

RC2119
: (2119, X'847') Application message data not converted.

On an MQGET call with the GMCONV option included in the *GMO* parameter, an error occurred during conversion of the data in the message. The message data is returned unconverted, the values of the *MDCSI* and *MDENC* fields in the

MSGDSC parameter are set to those of the message returned, and the call completes with CCWARN.

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 FMDLH), 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 error may also indicate that a parameter to the data-conversion service is not supported.

Corrective action: Check that the message data is correctly described by the *MDFMT*, *MDCSI* and *MDENC* parameters that were specified when the message was put. Also check that these values, and the *MDCSI* and *MDENC* specified in the *MSGDSC* 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.

RC2120
: (2120, X'848') Converted message too big for application buffer.

On an MQGET call, with the GMCONV option included in the *GMO* 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.

To avoid data being lost, the message is returned unconverted, with the *CMPCOD* parameter of the MQGET call set to CCWARN.

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 FMDLH), 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 *TGTBUF* 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 *DATLEN* parameter; in the case of a DBCS string or mixed SBCS/DBCS string, this length may be *less than* the length of *TGTBUF*.

Corrective action: For the MQGET call, check that the exit is converting the message data correctly and setting the output length *DATLEN* 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.

RC2125
: (2125, X'84D') Bridge started.

The IMS bridge has been started.

Corrective action: None. This reason code is only used to identify the corresponding event message.

RC2126
: (2126, X'84E') Bridge stopped.

The IMS bridge has been stopped.

Corrective action: None. This reason code is only used to identify the corresponding event message.

RC2135
: (2135, X'857') Distribution header structure not valid.

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, DOS client, HP-UX, OS/2, Sun Solaris, Windows client, Windows NT.

Corrective action: Correct the definition of the MQDH structure. Ensure that the fields are set correctly.

RC2136
: (2136, X'858') Multiple reason codes returned.

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 CCWARN in this case.
- The call failed for all of the destinations, but for differing reasons. The completion code is CCFAIL in this case.

This reason code occurs in the following environments: AIX, DOS client, HP-UX, OS/2, Sun Solaris, Windows client, Windows NT.

Corrective action: 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.

RC2137
: (2137, X'859') Queue not opened successfully.

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 *RRREA* field of the MQRR response record.

This reason code occurs in the following environments: AIX, DOS client, HP-UX, OS/2, Sun Solaris, Windows client, Windows NT.

Corrective action: 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.

RC2141
(2141, X'85D') Dead letter header structure not valid.

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, DOS client, HP-UX, OS/2, Sun Solaris, Windows client, Windows NT.

Corrective action: Correct the definition of the MQDLH structure. Ensure that the fields are set correctly.

RC2142
(2142, X'85E') MQ header structure not valid.

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, DOS client, HP-UX, OS/2, Sun Solaris, Windows client, Windows NT.

Corrective action: Correct the definition of the MQ header structure. Ensure that the fields are set correctly.

RC2143
(2143, X'85F') Source length parameter not valid.

On the MQXCNVC call, the *SRCLEN* 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 *SRCLEN* parameter pointer is not valid. (It is not always possible to detect an invalid parameter pointer; if not detected, unpredictable results occur.)

Corrective action: Specify a length that is zero or greater.

RC2144
(2144, X'860') Target length parameter not valid.

On the MQXCNVC call, the *TGTLEN* parameter specifies a length that is less than zero. This reason also occurs if the *TGTLEN* parameter pointer is not valid. (It is not always possible to detect an invalid parameter pointer; if not detected, unpredictable results occur.)

Corrective action: Specify a length that is zero or greater.

RC2145
(2145, X'861') Source buffer parameter not valid.

On the MQXCNVC call, the *SRCBUF* parameter pointer is not valid, or points to storage that cannot be accessed for the entire length specified by *SRCLEN*. (It is not always possible to detect an invalid parameter pointer; if not detected, unpredictable results occur.)

Corrective action: Specify a valid buffer.

RC2146
(2146, X'862') Target buffer parameter not valid.

On the MQXCNVC call, the *TGTBUF* parameter pointer is not valid, or points to read-only storage, or to storage that cannot be accessed for the entire length specified by *TGTLEN*. (It is not always possible to detect an invalid parameter pointer; if not detected, unpredictable results occur.)

Corrective action: Specify a valid buffer.

RC2148
(2148, X'864') IMS information header structure not valid.

On an MQPUT or MQPUT1 call, the IMS information header structure MQIIH in the message data is not valid.

Corrective action: Correct the definition of the MQIIH structure. Ensure that the fields are set correctly.

RC2149
(2149, X'865') PCF structures not valid.

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:

FMADMN
FMEVNT
FMPCF

Corrective action: 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.

RC2150
(2150, X'866') DBCS string not valid.

On the MQXCNVC call, the *SRCCSI* parameter specifies the coded character-set identifier of a double-byte character set (DBCS), but the *SRCBUF* parameter does not contain a valid DBCS string. This may be because the string contains characters which 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.

Corrective action: Specify a valid string.

RC2152
(2152, X'868') Object name not valid.

An MQOPEN or MQPUT1 call was issued to open a distribution list (that is, the *ODREC* field in MQOD is greater than zero), but the *ODON* field is neither blank nor the null string.

Corrective action: If it is intended to open a distribution list, set the *ODON* field to blanks or the null string. If it is not intended to open a distribution list, set the *ODREC* field to zero.

RC2153
(2153, X'869') Object queue-manager name not valid.

An MQOPEN or MQPUT1 call was issued to open a distribution list (that is, the *ODREC* field in MQOD is greater than zero), but the *ODMN* field is neither blank nor the null string.

Corrective action: If it is intended to open a distribution list, set the *ODMN* field to blanks or the null string. If it is not intended to open a distribution list, set the *ODREC* field to zero.

RC2154
(2154, X'86A') Number of records present not valid.

An MQOPEN or MQPUT1 call was issued, but the call failed for one of the following reasons:

- *ODREC* in MQOD is less than zero.
- *ODOT* in MQOD is not OTQ, and *ODREC* is not zero. *ODREC* must be zero if the object being opened is not a queue.

Corrective action: If it is intended to open a distribution list, set the *ODOT* field to OTQ and *ODREC* to the number of destinations in the list. If it is not intended to open a distribution list, set the *ODREC* field to zero.

RC2155
(2155, X'86B') Object records not valid.

An MQOPEN or MQPUT1 call was issued to open a distribution list (that is, the *ODREC* field in MQOD is greater than zero), but the MQOR object records are not specified correctly. One of the following applies:

- *ODORO* is zero and *ODORP* is the null pointer or zero.
- *ODORO* is not zero and *ODORP* is neither the null pointer nor zero.
- *ODORP* is not a valid pointer.
- *ODORP* or *ODORO* points to storage that is not accessible.

Corrective action: Ensure that one of *ODORO* and *ODORP* is zero and the other nonzero. Ensure that the field used points to accessible storage.

RC2156
(2156, X'86C') Response records not valid.

An MQOPEN or MQPUT1 call was issued to open a distribution list (that is, the *ODREC* field in MQOD is greater than zero), but the MQRR response records are not specified correctly. One of the following applies:

- *ODRRO* is not zero and *ODRRP* is neither the null pointer nor zero.
- *ODRRP* is not a valid pointer.
- *ODRRP* or *ODRRO* points to storage that is not accessible.

Corrective action: Ensure that at least one of *ODRRO* and *ODRRP* is zero. Ensure that the field used points to accessible storage.

RC2158
(2158, X'86E') Put message record flags not valid.

An MQPUT or MQPUT1 call was issued to put a message, but the *PMPRF* field in the MQPMO structure is not valid, for one of the following reasons:

- The field contains flags which are not valid.
- The message is being put to a distribution list, and put message records have been provided (that is, *PMREC* is greater than zero, and one of *PMPRO* or *PMPRP* is nonzero), but *PMPRF* has the value PFNONE.
- PFACC is specified without either PMSETI or PMSETA.

Corrective action: Ensure that *PMPRF* is set with the appropriate PF* flags to indicate which fields are present in the put message records. If PFACC is

specified, ensure that either PMSETI or PMSETA is also specified. Alternatively, set both *PMPRO* and *PMPRP* to zero.

RC2159
(2159, X'86F') Put message records not valid.

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:

- *PMPRO* is not zero and *PMPRP* is neither the null pointer nor zero.
- *PMPRP* is not a valid pointer.
- *PMPRP* or *PMPRO* points to storage that is not accessible.

Corrective action: Ensure that at least one of *PMPRO* and *PMPRP* is zero. Ensure that the field used points to accessible storage.

RC2161
(2161, X'871') Queue manager quiescing.

The application attempted to connect to the queue manager, but the queue manager is in the quiescing state.

On OS/400, the application either issued the MQCONN call, or issued the MQOPEN call when no connection was established.

This reason code also occurs if the queue manager is in the quiescing state and an application issues one of the following calls:

- MQOPEN, with OOFIQ included in the *OPTS* parameter
- MQGET, with GMFIQ included in the *GMOPT* field of the *GMO* parameter
- MQPUT or MQPUT1, with PMFIQ included in the *PMOPT* field of the *PMO* parameter

Corrective action: The application should tidy up and stop. If the OOFIQ, PMFIQ, and GMFIQ options are not used, the application may continue working in order to complete and commit the current unit of work; but it should not start another unit of work.

RC2162
(2162, X'872') Queue manager shutting down.

A call has been issued when the queue manager is shutting down. If the call is an MQGET call with the GMWT option, the wait has been canceled. No more message-queuing calls can be issued.

Corrective action: The application should tidy up and stop. Applications should ensure that any uncommitted updates are backed out; any unit of work that is coordinated by the queue manager is backed out automatically.

RC2173
(2173, X'87D') Put-message options structure not valid.

On an MQPUT or MQPUT1 call, the MQPMO structure is not valid. Either the *PMSID* mnemonic eye-catcher is not valid, or the *PMVER* is not recognized.

This reason also occurs if:

- The parameter pointer is not valid. (It is not always possible to detect an invalid parameter pointer; 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.

Corrective action: Correct the definition of the MQPMO structure. Ensure that required input fields are correctly set.

RC2184
(2184, X'888') Remote queue name not valid.

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 *ODMN* field in the object descriptor was not blank and not the name of the local queue manager, but the *ODON* field is blank.

Corrective action: Alter the local definition of the remote queue and supply a valid remote queue name, or supply a nonblank *ODON* in the object descriptor, as appropriate.

RC2185
(2185, X'889') Inconsistent persistence specification.

The MQPUT call was issued to put a message that has a value for the *MDPER* field in MQMD that is different from the previous message put using that queue handle. This is not permitted when the PMLOGO 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.

Corrective action: Modify the application to ensure that all of the messages in the group or logical message are put with the same value for the *MDPER* field in MQMD.

RC2186
(2186, X'88A') Get-message options structure not valid.

On an MQGET call, the MQGMO structure is not valid. Either the *GMSID* mnemonic eye-catcher is not valid, or the *GMVER* is not recognized.

This reason also occurs if:

- The parameter pointer is not valid. (It is not always possible to detect an invalid parameter pointer; 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.

Corrective action: Correct the definition of the MQGMO structure. Ensure that required input fields are correctly set.

RC2187
(2187, X'88B') Requested function not supported by CICS bridge.

It is not permitted to use the MQI from user transactions that are run in an MQSeries-CICS bridge environment where the bridge exit also uses the MQI. The MQI request 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.

Corrective action: 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.

RC2191
: (2191, X'88F') Character trigger message structure not valid.

On an MQPUT or MQPUT1 call, the character trigger message structure MQTMC in the message data is not valid.

Corrective action: Correct the definition of the MQTMC structure. Ensure that the fields are set correctly.

RC2194
: (2194, X'892') Object name not valid for object type.

An MQOPEN call was issued to open the queue manager definition, but the *ODON* field in the *OBJDSC* parameter is not blank.

Corrective action: Ensure that the *ODON* field is set to blanks.

RC2195
: (2195, X'893') Unexpected error occurred.

The call was rejected because an unexpected error occurred.

Corrective Action: 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.

Consult the FFST record to obtain more detail about the problem.

RC2196
: (2196, X'894') Unknown transmission queue.

On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager. The *ODON* or the *ODMN* 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.

Corrective action: Check the values specified for *ODON* and *ODMN*. If these are correct, check the queue definitions. For more information on transmission queues, see the *MQSeries Application Programming Guide*.

RC2197
: (2197, X'895') Unknown default transmission queue.

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.

Corrective Action: Correct the queue definitions, or the queue-manager attribute. See the *MQSeries Application Programming Guide* for more information.

RC2198
(2198, X'896') Default transmission queue not local.

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.

Corrective Action: 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 which 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.

RC2199
(2199, X'897') Default transmission queue usage error.

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 USTRAN.

Corrective Action: 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 which 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 USTRAN.

See the *MQSeries Application Programming Guide* for more information.

RC2209
(2209, X'8A1') No message locked.

An MQGET call was issued with the GMUNLK option, but no message was currently locked.

Corrective action: Check that a message was locked by an earlier MQGET call with the GMLK option for the same handle, and that no intervening call has caused the message to become unlocked.

RC2218
: (2218, X'8AA') Message length greater than maximum for channel.

  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 *MDFB* field in the message descriptor of a report message.

  Corrective action: Check the channel definitions. Increase the maximum message length that the channel can accept, or break the message into several smaller messages.

RC2219
: (2219, X'8AB') MQI call reentered before previous call complete.

  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.

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

RC2220
: (2220, X'8AC') Reference message header structure not valid.

  On an MQPUT or MQPUT1 call, the reference message header structure MQRMH in the message data is not valid.

  Corrective action: Correct the definition of the MQRMH structure. Ensure that the fields are set correctly.

RC2222
: (2222, X'8AE') Queue manager created.

  This condition is detected when a queue manager becomes active.

  Corrective action: None. This reason code is only used to identify the corresponding event message.

RC2223
: (2223, X'8AE') Queue manager unavailable.

  This condition is detected when a queue manager is requested to stop or quiesce.

  Corrective action: None. This reason code is only used to identify the corresponding event message.

RC2224
: (2224, X'8B0') Queue depth high limit reached or exceeded.

  An MQPUT or MQPUT1 call has caused the queue depth to be incremented to or above the limit specified in the *QDepthHighLimit* attribute.

  Corrective action: None. This reason code is only used to identify the corresponding event message.

RC2225
: (2225, X'8B1') Queue depth low limit reached or exceeded.

An MQGET call has caused the queue depth to be decremented to or below the limit specified in the *QDepthLowLimit* attribute.

Corrective action: None. This reason code is only used to identify the corresponding event message.

RC2226
: (2226, X'8B2') Queue service interval high.

No successful gets or puts have been detected within an interval which is greater than the limit specified in the *QServiceInterval* attribute.

Corrective action: None. This reason code is only used to identify the corresponding event message.

RC2227
: (2227, X'8B3') Queue service interval ok.

A successful get has been detected within an interval which is less than or equal to the limit specified in the *QServiceInterval* attribute.

Corrective action: None. This reason code is only used to identify the corresponding event message.

RC2233
: (2233, X'8B9') Automatic channel definition succeeded.

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

Corrective action: None. This reason code is only used to identify the corresponding event message.

RC2234
: (2234, X'8BA') Automatic channel definition failed.

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.

Corrective action: Examine the additional information returned in the event message to determine the reason for the failure.

RC2235
: (2235, X'8BB') PCF header structure not valid.

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, DOS client, HP-UX, OS/2, Sun Solaris, Windows client, Windows NT.

Corrective action: Correct the definition of the MQCFH structure. Ensure that the fields are set correctly.

RC2236
: (2236, X'8BC') PCF integer list parameter structure not valid.

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

This reason code occurs in the following environments: AIX, DOS client, HP-UX, OS/2, Sun Solaris, Windows client, Windows NT.

Corrective action: Correct the definition of the MQCFIL structure. Ensure that the fields are set correctly.

RC2237
(2237, X'8BD') PCF integer parameter structure not valid.

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

This reason code occurs in the following environments: AIX, DOS client, HP-UX, OS/2, Sun Solaris, Windows client, Windows NT.

Corrective action: Correct the definition of the MQCFIN structure. Ensure that the fields are set correctly.

RC2238
(2238, X'8BE') PCF string list parameter structure not valid.

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

This reason code occurs in the following environments: AIX, DOS client, HP-UX, OS/2, Sun Solaris, Windows client, Windows NT.

Corrective action: Correct the definition of the MQCFSL structure. Ensure that the fields are set correctly.

RC2239
(2239, X'8BF') PCF string parameter structure not valid.

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

This reason code occurs in the following environments: AIX, DOS client, HP-UX, OS/2, Sun Solaris, Windows client, Windows NT.

Corrective action: Correct the definition of the MQCFST structure. Ensure that the fields are set correctly.

RC2241
(2241, X'8C1') Message group not complete.

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 which is not in a group and specifies PMLOGO. The call fails in this case.
- On the MQPUT call, when the application attempts to put a message which is not the next one in the group, does *not* specify PMLOGO, but the previous MQPUT call for the queue handle did specify PMLOGO. The call succeeds with completion code CCWARN in this case.
- On the MQGET call, when the application attempts to get a message which is not the next one in the group, does *not* specify GMLOGO, but the previous MQGET call for the queue handle did specify GMLOGO. The call succeeds with completion code CCWARN 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 CCWARN.

If there is an incomplete logical message as well as an incomplete message group, reason code RC2242 is returned in preference to RC2241.

Corrective action: 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 MFLMIG.

RC2242
(2242, X'8C2') Logical message not complete.

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 which is not a segment and specifies PMLOGO. The call fails in this case.
- On the MQPUT call, when the application attempts to put a message which is not the next segment, does *not* specify PMLOGO, but the previous MQPUT call for the queue handle did specify PMLOGO. The call succeeds with completion code CCWARN in this case.
- On the MQGET call, when the application attempts to get a message which is not the next segment, does *not* specify GMLOGO, but the previous MQGET call for the queue handle did specify GMLOGO. The call succeeds with completion code CCWARN 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 CCWARN.

Corrective action: If this reason code is expected, no corrective action is required. Otherwise, ensure that the MQPUT call for the last segment specifies MFLSEG.

RC2243
(2243, X'8C3') Message segments have differing CCSIDs.

An MQGET call was issued specifying the GMCMPM option, but the message to be retrieved consists of two or more segments which have differing values for the *MDCSI* 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 CCWARN, but only the first few segments that have identical character-set identifiers are returned.

Corrective action: Remove the GMCMPM option from the MQGET call and retrieve the remaining message segments one by one.

RC2244
(2244, X'8C4') Message segments have differing encodings.

An MQGET call was issued specifying the GMCMPM option, but the message to be retrieved consists of two or more segments which have differing values for the *MDENC* 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 CCWARN, but only the first few segments that have identical encodings are returned.

Corrective action: Remove the GMCMPM option from the MQGET call and retrieve the remaining message segments one by one.

RC2245
: (2245, X'8C5') Inconsistent unit-of-work specification.

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 PMSYP 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 PMLOGO, the call fails. If the current call does not specify PMLOGO, but the previous MQPUT call for the queue handle did, the call succeeds with completion code CCWARN.

- 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 GMSYP 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 GMLOGO, the call fails. If the current call does not specify GMLOGO, but the previous MQGET call for the queue handle did, the call succeeds with completion code CCWARN.

Corrective action: 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.

RC2246
: (2246, X'8C6') Message under cursor not valid for retrieval.

An MQGET call was issued specifying the GMCMPM option with either GMMUC or GMBRWC, but the message that is under the cursor has an MQMD with an *MDOFF* field that is greater than zero. Because GMCMPM was specified, the message is not valid for retrieval.

Corrective action: Reposition the browse cursor so that it is located on a message whose *MDOFF* field in MQMD is zero. Alternatively, remove the GMCMPM option.

RC2247
: (2247, X'8C7') Match options not valid.

An MQGET call was issued, but the value of the *GMMO* field in the *GMO* parameter is not valid. Either an undefined option is specified, or a defined option which is not valid in the current circumstances is specified. In the latter case, it means that all of the following are true:

- GMLOGO is specified.
- There is a current message group or logical message for the queue handle.
- Neither of the following options is specified:

  GMBRWC
  GMMUC

- One or more of the MO* options is specified.
- The values of the fields in the *MSGDSC* parameter corresponding to the MO* options specified, differ from the values of those fields in the MQMD for the message to be returned next.

Corrective action: Ensure that only valid options are specified for the field.

RC2248
(2248, X'8C8') Message descriptor extension not valid.

The MQMDE structure at the start of the application message data is not valid, for one of the following reasons:

- The *MESID* mnemonic eye-catcher is not MESIDV.
- The *MEVER* field is less than MEVER2.
- The *MELEN* field is less than MELEN2, or (for *MEVER* equal to MEVER2 only) greater than MELEN2.

Corrective action: Correct the definition of the message descriptor extension. Ensure that required input fields are correctly set.

RC2249
(2249, X'8C9') Message flags not valid.

An MQPUT or MQPUT1 call was issued, but the *MDMFL* field in the message descriptor MQMD contains one or more message flags which 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 C, "Report options" on page 379 for more details.

This reason code can also occur in the *MDFB* field in the MQMD of a report message, or in the *DLREA* 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.

Corrective action: Do the following:

1. Ensure that the *MDMFL* 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 MFNONE if no message flags are needed.

2. Ensure that the message flags specified are ones which are documented in this book; see the *MDMFL* field described in "MQMD – Message descriptor" on page 59 for valid message flags. Remove any message flags which are not documented in this book.
3. If multiple message flags are being set by adding the individual message flags together, ensure that the same message flag is not added twice.

RC2250
(2250, X'8CA') Message sequence number not valid.

An MQPUT or MQPUT1 call was issued, but the value of the *MDSEQ* 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 *MDSEQ* field would have become greater than 999 999 999 as a result of the call.

Corrective action: 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.

RC2251
: (2251, X'8CB') Message segment offset not valid.

An MQPUT or MQPUT1 call was issued, but the value of the *MDOFF* 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 *MDOFF* field would have become greater than 999 999 999 as a result of the call.

Corrective action: Specify a value in the range 0 through 999 999 999. Do not attempt to create a message segment which would extend beyond an offset of 999 999 999.

RC2252
: (2252, X'8CC') Original length not valid.

An MQPUT or MQPUT1 call was issued to put a report message which is reporting on a segment, but the *MDOLN* field in the MQMD or MQMDE structure is either:

- Less than one (for a segment which is not the last segment), or
- Less than zero (for a segment which is the last segment)

Corrective action: Specify a value which is greater than zero. Zero is valid only for the last segment.

RC2253
: (2253, X'8CD') Length of data in message segment is zero.

An MQPUT or MQPUT1 call was issued to put the first or 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.

Corrective action: 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 zero length.

RC2255
: (2255, X'8CF') Unit of work not available for the queue manager to use.

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 GMCMPM 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 MFSEGA flag is specified in MQMD and the message requires segmentation.

Corrective action: 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.

RC2256

(2256, X'8D0') Wrong version of MQGMO supplied.

An MQGET call was issued specifying options that required an MQGMO with a version number not less than GMVER2, but the MQGMO supplied did not satisfy this condition.

Corrective action: Modify the program to pass a version-2 MQGMO. Check the program logic to ensure that the *GMVER* field in MQGMO has been set to GMVER2. Alternatively, remove the option that requires the version-2 MQGMO.

RC2257

(2257, X'8D1') Wrong version of MQMD supplied.

An MQGET, MQPUT, or MQPUT1 call was issued specifying options that required an MQMD with a version number not less than MDVER2, but the MQMD supplied did not satisfy this condition.

Corrective action: Modify the program to pass a version-2 MQMD. Check the program logic to ensure that the *MDVER* field in MQMD has been set to MDVER2. Alternatively, remove the option that requires the version-2 MQMD.

RC2258

(2258, X'8D2') Group identifier not valid.

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:

- PMLOGO is not specified in the *PMOPT* field in MQPMO.
- Either there are too few MQPMR records provided by MQPMO, or the *PRGID* field is not present in the MQPMR records.
- One or more of the following flags is specified in the *MDMFL* field in MQMD or MQMDE:

  MFSEGA
  MF*MIG
  MF*SEG
- The *MDGID* field in MQMD or MQMDE is not GINONE.

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.

Corrective action: Specify GINONE for the *MDGID* field in MQMD or MQMDE. Alternatively, if the call is MQPUT specify PMLOGO in the *PMOPT* field in MQPMO.

RC2259

(2259, X'8D3') Inconsistent browse specification.

An MQGET call was issued with the GMBRWN option specified, but the specification of the GMLOGO 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 GMLOGO, or neither call must specify GMLOGO.

Corrective action: Add or remove the GMLOGO option as appropriate. Alternatively, to switch between logical order and physical order, specify the GMBRWF option to restart the scan from the beginning of the queue, and either omit or specify GMLOGO as desired.

RC2260
(2260, X'8D4') Transmission queue header structure not valid.

On an MQPUT or MQPUT1 call, the transmission queue header structure MQXQH in the message data is not valid.

Corrective action: Correct the definition of the MQXQH structure. Ensure that the fields are set correctly.

RC2261
(2261, X'8D5') Source environment data error.

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:

- *RMSEL* is less than zero.
- Source environment data is not present although *RMSEL* is greater than zero.
- The range defined by *RMSEO* and *RMSEL* is not wholly beyond the fixed fields in the MQRMH structure and within *RMLEN* bytes from the start of the structure.

The exit returns this reason in the *CXFB* field of the MQCXP structure. If an exception report is requested, it is copied to the *CXFB* field of the MQMD associated with the report.

Corrective action: Specify the source environment data correctly.

RC2262
(2262, X'8D6') Source name data error.

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:

- *RMSNL* is less than zero.
- Source name data is not present although *RMSNL* is greater than zero.
- The range defined by *RMSNO* and *RMSNL* is not wholly beyond the fixed fields in the MQRMH structure and within *RMLEN* bytes from the start of the structure.

The exit returns this reason in the *CXFB* field of the MQCXP structure. If an exception report is requested, it is copied to the *CXFB* field of the MQMD associated with the report.

Corrective action: Specify the source name data correctly.

RC2263
(2263, X'8D7') Destination environment data error.

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:

- *RMDEL* is less than zero.

- Destination environment data is not present although *RMDEL* is greater than zero.
- The range defined by *RMDEO* and *RMDEL* is not wholly beyond the fixed fields in the MQRMH structure and within *RMLEN* bytes from the start of the structure.

The exit returns this reason in the *CXFB* field of the MQCXP structure. If an exception report is requested, it is copied to the *CXFB* field of the MQMD associated with the report.

Corrective action: Specify the destination environment data correctly.

RC2264
: (2264, X'8D8') Destination name data error.

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:

- *RMDNL* is less than zero.
- Destination name data is not present although *RMDNL* is greater than zero.
- The range defined by *RMDNO* and *RMDNL* is not wholly beyond the fixed fields in the MQRMH structure and within *RMLEN* bytes from the start of the structure.

The exit returns this reason in the *CXFB* field of the MQCXP structure. If an exception report is requested, it is copied to the *CXFB* field of the MQMD associated with the report.

Corrective action: Specify the destination name data correctly.

RC2265
: (2265, X'8D9') Trigger message structure not valid.

On an MQPUT or MQPUT1 call, the trigger message structure MQTM in the message data is not valid.

Corrective action: Correct the definition of the MQTM structure. Ensure that the fields are set correctly.

RC2282
: (2282, X'8EA') Channel started.

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.

Corrective action: None. This reason code is only used to identify the corresponding event message.

RC2283
: (2283, X'8EB') Channel stopped.

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

Corrective action: None. This reason code is only used to identify the corresponding event message.

RC2284
(2284, X'8EC') Channel conversion error.

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.

Corrective action: None. This reason code is only used to identify the corresponding event message.

RC2295
(2295, X'8F7') Channel activated.

This condition is detected when a channel which 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 which is able to become active without waiting for an active slot to be released.

Corrective action: None. This reason code is only used to identify the corresponding event message.

RC2296
(2296, X'8F8') Channel cannot be activated.

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. 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.

Corrective action: None. This reason code is only used to identify the corresponding event message.

Chapter 6. MQSeries constants

Note

The names of the MQI constants are listed in this chapter in the form in which they appear in the RPG COPY file. The copy file is named CMQR.

This chapter specifies the values of all of the named constants that are mentioned in this book. For other MQI constants, refer to the *MQSeries Intercommunication* and *MQSeries Programmable System Management*.

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 "XX" that indicates the nature of the values defined in that group. The constants are ordered alphabetically by the 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".

List of constants

The following sections list all of the named constants mentioned in this book, and show their values.

LN* (Lengths of character string and byte fields)

See the *CHRATR* parameter described in "MQINQ – Inquire about object attributes" on page 194 and "MQSET – Set object attributes" on page 236.

LNABNC	4	X'00000004'
LNACCT	32	X'00000020'
LNAIDD	32	X'00000020'
LNAORD	4	X'00000004'
LNAPPN	28	X'0000001C'
LNATID	4	X'00000004'
LNAUTH	8	X'00000008'
LNBRGN	24	X'00000018'
LNCDAT	12	X'0000000C'
LNCHD	64	X'00000040'
LNCHN	20	X'00000014'
LNCID	24	X'00000018'
LNCNCL	4	X'00000004'

© Copyright IBM Corp. 1994, 1998

LNCONN	264	X'00000108'
LNCRTD	12	X'0000000C'
LNCRTT	8	X'00000008'
LNCTIM	8	X'00000008'
LNEXDA	32	X'00000020'
LNEXN	20	X'00000014'
LNEXUA	16	X'00000010'
LNFAC	8	X'00000008'
LNFACL	4	X'00000004'
LNFMT	8	X'00000008'
LNFUNC	4	X'00000004'
LNGID	24	X'00000018'
LNLTOV	8	X'00000008'
LNLUWI	16	X'00000010'
LNMCAJ	28	X'0000001C'
LNMCAN	20	X'00000014'
LNMFMN	8	X'00000008'
LNMHD	4000	X'00000FA0'
LNMID	24	X'00000018'
LNMODN	8	X'00000008'
LNNLD	64	X'00000040'
LNNLN	48	X'00000030'
LNOIID	24	X'00000018'
LNPAN	28	X'0000001C'
LNPDAT	8	X'00000008'
LNPROA	256	X'00000100'
LNPROD	64	X'00000040'
LNPROE	128	X'00000080'
LNPRON	48	X'00000030'
LNPROU	128	X'00000080'
LNPTIM	8	X'00000008'
LNPWRD	12	X'0000000C'
LNQD	64	X'00000040'
LNQMD	64	X'00000040'
LNQMN	48	X'00000030'
LNQN	48	X'00000030'
LNRSID	4	X'00000004'
LNSCON	20	X'00000014'
LNSTCO	4	X'00000004'
LNSTGC	8	X'00000008'
LNTEXD	999	X'000003E7'
LNTEXN	999	X'000003E7'
LNTIID	16	X'00000010'
LNTPN	64	X'00000040'
LNTRGD	64	X'00000040'
LNTRID	4	X'00000004'
LNUID	12	X'0000000C'

AC* (Accounting token)

See the *MDACC* field described in “MQMD – Message descriptor” on page 59.

ACNONE	X'00...00' (32 nulls)

AT* (Application type)

See the *MDPAT* field described in “MQMD – Message descriptor” on page 59, and the *ApplType* attribute described in “Attributes for process definitions” on page 262.

ATUNK	-1	X'FFFFFFFF'
ATNCON	0	X'00000000'
ATCICS	1	X'00000001'
ATMVS	2	X'00000002'
ATIMS	3	X'00000003'
ATOS2	4	X'00000004'
ATDOS	5	X'00000005'
ATAIX	6	X'00000006'
ATUNIX	6	X'00000006'
ATQM	7	X'00000007'
AT400	8	X'00000008'
ATDEF	8	X'00000008'
ATWIN	9	X'00000009'
ATVSE	10	X'0000000A'
ATWINT	11	X'0000000B'
ATVMS	12	X'0000000C'
ATGUAR	13	X'0000000D'
ATVOS	14	X'0000000E'
ATIMSB	19	X'00000013'
ATXCF	20	X'00000014'
ATCICB	21	X'00000015'
ATUFST	65536	X'00010000'
ATULST	999999999	X'3B9AC9FF'

CA* (Character attribute selector)

See the *SELS* parameter described in “MQINQ – Inquire about object attributes” on page 194 and “MQSET – Set object attributes” on page 236.

CAFRST	2001	X'000007D1'
CAAPPI	2001	X'000007D1'
CABASQ	2002	X'000007D2'
CACMDQ	2003	X'000007D3'
CACRTD	2004	X'000007D4'
CACRTT	2005	X'000007D5'
CADLQ	2006	X'000007D6'
CAENVD	2007	X'000007D7'
CAINIQ	2008	X'000007D8'
CALSTD	2009	X'000007D9'
CALSTN	2010	X'000007DA'
CAPROD	2011	X'000007DB'
CAPRON	2012	X'000007DC'
CAQD	2013	X'000007DD'
CAQMD	2014	X'000007DE'

CAQMN	2015	X'000007DF'
CAQN	2016	X'000007E0'
CARQMN	2017	X'000007E1'
CARQN	2018	X'000007E2'
CABRQN	2019	X'000007E3'
CANAMS	2020	X'000007E4'
CAUSRD	2021	X'000007E5'
CASTGC	2022	X'000007E6'
CATRGD	2023	X'000007E7'
CAXQN	2024	X'000007E8'
CADXQN	2025	X'000007E9'
CACADX	2026	X'000007EA'
CALSTU	2026	X'000007EA'
CALAST	4000	X'00000FA0'

CC* (Completion code)

See the *CMPCOD* parameter.

CCUNK	-1	X'FFFFFFFF'
CCOK	0	X'00000000'
CCWARN	1	X'00000001'
CCFAIL	2	X'00000002'

CS* (Coded character set identifier)

See the *MDCSI* field described in "MQMD – Message descriptor" on page 59.

CSEMBD	-1	X'FFFFFFFF'
CSDEF	0	X'00000000'
CSQM	0	X'00000000'

CI* (Correlation identifier)

See the *MDCID* field described in "MQMD – Message descriptor" on page 59.

CINONE	X'00...00' (24 nulls)
CINEWS	X'414D51214E45575F534553...'

MQ* (Call identifier)

MQCONN	1	X'00000001'
MQDISC	2	X'00000002'
MQOPEN	3	X'00000003'
MQCLOS	4	X'00000004'
MQGET	5	X'00000005'
MQPUT	6	X'00000006'
MQPUT1	7	X'00000007'
MQINQ	8	X'00000008'
MQSET	9	X'00000009'
MQXCVC	12	X'0000000C'

CMLV* (Command level)

See the *CommandLevel* attribute described in “Attributes for the queue-manager” on page 264.

CMLVL1	100	X'00000064'
CML101	101	X'00000065'
CML110	110	X'0000006E'
CML114	114	X'00000072'
CML120	120	X'00000078'
CML200	200	X'000000C8'
CML201	201	X'000000C9'
CML221	221	X'000000DD'
CML320	320	X'00000140'
CML420	420	X'000001A4'
CML500	500	X'000001F4'

CO* (Close options)

See the *OPTS* parameter described in “MQCLOSE – Close object” on page 169.

CONONE	0	X'00000000'
CODEL	1	X'00000001'
COPURG	2	X'00000002'

DCC* (Convert-characters masks and factors)

See the *OPTS* parameter described in “MQXCNVC - Convert characters” on page 403.

DCCSMA	240	X'000000F0'
DCCTMA	3840	X'00000F00'
DCCSFA	16	X'00000010'
DCCTFA	256	X'00000100'

DCC* (Convert-characters options)

See the *OPTS* parameter described in “MQXCNVC - Convert characters” on page 403.

DCCSUN	0	X'00000000'
DCCTUN	0	X'00000000'
DCCNON	0	X'00000000'
DCCDEF	1	X'00000001'
DCCSNA	16	X'00000010'
DCCSNO	16	X'00000010'
DCCSRE	32	X'00000020'
DCCTNA	256	X'00000100'
DCCTNO	256	X'00000100'
DCCTRE	512	X'00000200'

DH* (Distribution header structure identifier)

See the *DHSID* field described in "MQDH – Distribution header" on page 13.

DHSIDV	'DHbb'	

DH* (Distribution header version)

See the *DHVER* field described in "MQDH – Distribution header" on page 13.

DHVER1	1	X'00000001'
DHVERC	1	X'00000001'

DHF* (Distribution header flags)

See the *DHFLG* field described in "MQDH – Distribution header" on page 13.

DHFNON	0	X'00000000'
DHFNEW	1	X'00000001'

DL* (Distribution list support)

See the *DistLists* attributes described in "Attributes for the queue-manager" on page 264 and "Attributes for local queues" on page 246.

DLNSUP	0	X'00000000'
DLSUPP	1	X'00000001'

DL* (Dead-letter header structure identifier)

See the *DLSID* field described in "MQDLH – Dead-letter (undelivered-message) header" on page 19.

DLSIDV	'DLHb'	

DL* (Dead-letter header version)

See the *DLVER* field described in "MQDLH – Dead-letter (undelivered-message) header" on page 19.

DLVER1	1	X'00000001'
DLVERC	1	X'00000001'

DX* (Data-conversion-exit parameter structure identifier)

See the *DXSID* field described in "MQDXP - Data conversion header" on page 391.

DXSIDV	'DXPb'	

DX* (Data-conversion-exit parameter structure version)

See the *DXVER* field described in "MQDXP - Data conversion header" on page 391.

DXVER1	1	X'00000001'
DXVERC	1	X'00000001'

EI* (Expiry interval)

See the *MDEXP* field described in "MQMD – Message descriptor" on page 59.

EIULIM	-1	X'FFFFFFFF'

EN* (Encoding)

See the *MDENC* field described in "MQMD – Message descriptor" on page 59.

ENNAT	273	X'00000111'

EN* (Encoding masks)

See Appendix B, "Machine encodings" on page 375.

ENIMSK	15	X'0000000F'
ENDMSK	240	X'000000F0'
ENFMSK	3840	X'00000F00'
ENRMSK	-4096	X'FFFFF000'

EN* (Encoding for packed-decimal integers)

See Appendix B, "Machine encodings" on page 375.

ENDUND	0	X'00000000'
ENDNOR	16	X'00000010'
ENDREV	32	X'00000020'

EN* (Encoding for floating-point numbers)

See Appendix B, "Machine encodings" on page 375.

ENFUND	0	X'00000000'
ENFNOR	256	X'00000100'
ENFREV	512	X'00000200'
ENF390	768	X'00000300'

EN* (Encoding for binary integers)

See Appendix B, "Machine encodings" on page 375.

ENIUND	0	X'00000000'
ENINOR	1	X'00000001'
ENIREV	2	X'00000002'

EV* (Event reporting)

EVRDIS	0	X'00000000'
EVRENA	1	X'00000001'

FB* (Feedback)

See the *MDFB* field described in "MQMD – Message descriptor" on page 59, and the *DLREA* field described in "MQDLH – Dead-letter (undelivered-message) header" on page 19; see also the RC* values.

FBNONE	0	X'00000000'

FBSFST	1	X'00000001'
FBQUIT	256	X'00000100'
FBEXP	258	X'00000102'
FBCOA	259	X'00000103'
FBCOD	260	X'00000104'
FBCHNC	262	X'00000106'
FBCHNR	263	X'00000107'
FBCHNF	264	X'00000108'
FBABEG	265	X'00000109'
FBTM	266	X'0000010A'
FBATYP	267	X'0000010B'
FBSBMX	268	X'0000010C'
FBXQME	271	X'0000010F'
FBPAN	275	X'00000113'
FBNAN	276	X'00000114'
FBDLZ	291	X'00000123'
FBDLN	292	X'00000124'
FBDLTB	293	X'00000125'
FBBUFO	294	X'00000126'
FBLOB1	295	X'00000127'
FBIIH	296	X'00000128'
FBNAFI	298	X'0000012A'
FBIERR	300	X'0000012C'
FBIFST	301	X'0000012D'
FBILST	399	X'0000018F'
FBCINE	401	X'00000191'
FBCNTA	402	X'00000192'
FBCBRF	403	X'00000193'
FBCCIE	404	X'00000194'
FBCCSE	405	X'00000195'
FBCENE	406	X'00000196'
FBCIHE	407	X'00000197'
FBCUWE	408	X'00000198'
FBCCAE	409	X'00000199'
FBCANS	410	X'0000019A'
FBCAAB	411	X'0000019B'
FBCDLQ	412	X'0000019C'
FBCUBO	413	X'0000019D'
FBSLST	65535	X'0000FFFF'
FBAFST	65536	X'00010000'
FBALST	999999999	X'3B9AC9FF'

FM* (Format)

See the *MDFMT* field described in “MQMD – Message descriptor” on page 59.

FMNONE	'bbbbbbbb'
FMADMN	'MQADMINb'
FMCHNC	'MQCHCOMb'
FMCMD1	'MQCMD1bb'
FMCMD2	'MQCMD2bb'
FMDLH	'MQDEADbb'
FMDH	'MQHDISTb'
FMEVNT	'MQEVENTb'

FMIMS	'MQIMSbbb'
FMIMVS	'MQIMSVSb'
FMMDE	'MQHMDEbb'
FMPCF	'MQPCFbbb'
FMRMH	'MQHREFbb'
FMSTR	'MQSTRbbb'
FMTM	'MQTRIGbb'
FMXQH	'MQXMITbb'

GI* (Group identifier)

See the *MDGID* field described in "MQMD – Message descriptor" on page 59.

GINONE	X'00...00' (24 nulls)

GM* (Get message options)

See the *GMOPT* field described in "MQGMO – Get message options" on page 27.

GMNWT	0	X'00000000'
GMNONE	0	X'00000000'
GMWT	1	X'00000001'
GMSYP	2	X'00000002'
GMNSYP	4	X'00000004'
GMBRWF	16	X'00000010'
GMBRWN	32	X'00000020'
GMATM	64	X'00000040'
GMMUC	256	X'00000100'
GMLK	512	X'00000200'
GMUNLK	1024	X'00000400'
GMBRWC	2048	X'00000800'
GMPSYP	4096	X'00001000'
GMFIQ	8192	X'00002000'
GMCONV	16384	X'00004000'
GMLOGO	32768	X'00008000'
GMCMPM	65536	X'00010000'
GMAMSA	131072	X'00020000'
GMASGA	262144	X'00040000'

GM* (Get message options structure identifier)

See the *GMSID* field described in "MQGMO – Get message options" on page 27.

GMSIDV	'GMOb'

GM* (Get message options version)

See the *GMVER* field described in "MQGMO – Get message options" on page 27.

GMVER1	1	X'00000001'
GMVER2	2	X'00000002'
GMVERC	2	X'00000002'

GS* (Group status)

See the *GMGST* field described in "MQGMO – Get message options" on page 27.

GSNIG	'b'	
GSMIG	'G'	
GSLMIG	'L'	

HC* (Connection handle)

See the *HCONN* parameter described in "MQCONN – Connect queue manager" on page 175 and "MQDISC – Disconnect queue manager" on page 180.

HCUNUH	-1	X'FFFFFFFF'
HCDEFH	0	X'00000000'

HO* (Object handle)

See the *HOBJ* parameter described in "MQCLOSE – Close object" on page 169.

HOUNUH	-1	X'FFFFFFFF'

IA* (Integer attribute selector)

See the *SELS* parameter described in "MQINQ – Inquire about object attributes" on page 194 and "MQSET – Set object attributes" on page 236.

IAFRST	1	X'00000001'
IAAPPT	1	X'00000001'
IACCSI	2	X'00000002'
IACDEP	3	X'00000003'
IADINP	4	X'00000004'
IADPER	5	X'00000005'
IADPRI	6	X'00000006'
IADEFT	7	X'00000007'
IAHGB	8	X'00000008'
IAIGET	9	X'00000009'
IAIPUT	10	X'0000000A'
IAMHND	11	X'0000000B'
IAUSAG	12	X'0000000C'
IAMLEN	13	X'0000000D'
IAMPRI	14	X'0000000E'
IAMDEP	15	X'0000000F'
IAMDS	16	X'00000010'
IAOIC	17	X'00000011'
IAOOC	18	X'00000012'
IANAMC	19	X'00000013'
IAQTYP	20	X'00000014'
IARINT	21	X'00000015'
IABTHR	22	X'00000016'
IASHAR	23	X'00000017'
IATRGC	24	X'00000018'
IATRGI	25	X'00000019'
IATRGP	26	X'0000001A'
IATRGT	28	X'0000001C'
IATRGD	29	X'0000001D'

IASYNC	30	X'0000001E'
IACMDL	31	X'0000001F'
IAPLAT	32	X'00000020'
IAMUNC	33	X'00000021'
IADIST	34	X'00000022'
IATSR	35	X'00000023'
IAHQD	36	X'00000024'
IAMEC	37	X'00000025'
IAMDC	38	X'00000026'
IAQDHL	40	X'00000028'
IAQDLL	41	X'00000029'
IAQDME	42	X'0000002A'
IAQDHE	43	X'0000002B'
IAQDLE	44	X'0000002C'
IASCOP	45	X'0000002D'
IAQSIE	46	X'0000002E'
IAAUTE	47	X'0000002F'
IAINHE	48	X'00000030'
IALCLE	49	X'00000031'
IARMTE	50	X'00000032'
IASSE	52	X'00000034'
IAPFME	53	X'00000035'
IAQSI	54	X'00000036'
IACAD	55	X'00000037'
IACADE	56	X'00000038'
IALSTU	56	X'00000038'
IAINDT	57	X'00000039'
IALAST	2000	X'000007D0'

IAU* (IMS authenticator)

See the *IIAUT* field described in "MQIIH – IMS header" on page 53.

IAUNON	'bbbbbbbb'

IAV* (Integer attribute value)

See the *INTATR* parameter described in "MQINQ – Inquire about object attributes" on page 194.

IAVUND	-2	X'FFFFFFFE'
IAVNA	-1	X'FFFFFFFF'

ICM* (IMS commit mode)

See the *IICMT* field described in "MQIIH – IMS header" on page 53.

ICMCTS	'0'
ICMSTC	'1'

II* (IMS header flags)

See the *IIFLG* field described in "MQIIH – IMS header" on page 53.

IINONE	0	X'00000000'

II* (IMS header length)

See the *IILEN* field described in "MQIIH – IMS header" on page 53.

IILEN1	84	X'00000054'

II* (IMS header structure identifier)

See the *IISID* field described in "MQIIH – IMS header" on page 53.

IISIDV	'IIHb'

II* (IMS header version)

See the *IIVER* field described in "MQIIH – IMS header" on page 53.

IIVER1	1	X'00000001'
IIVERC	1	X'00000001'

ISS* (IMS security scope)

See the *IISEC* field described in "MQIIH – IMS header" on page 53.

ISSCHK	'C'
ISSFUL	'F'

IT* (Index type)

See the *IndexType* attribute described in "Attributes for local queues" on page 246.

ITNONE	0
ITMSGI	1
ITCORI	2

ITI* (IMS transaction instance identifier)

See the *IITID* field described in "MQIIH – IMS header" on page 53.

ITINON	X'00...00' (16 nulls)

ITS* (IMS transaction state)

See the *IITST* field described in "MQIIH – IMS header" on page 53.

ITSIC	'C'
ITSNIC	' '

MD* (Message descriptor structure identifier)

See the *MDSID* field described in “MQMD – Message descriptor” on page 59.

MDSIDV	'MDbb'	

MD* (Message descriptor version)

See the *MDVER* field described in “MQMD – Message descriptor” on page 59.

MDVER1	1	X'00000001'
MDVER2	2	X'00000002'
MDVERC	2	X'00000002'

ME* (Message descriptor extension length)

See the *MELEN* field described in “MQMDE – Message descriptor extension” on page 104.

MELEN2	72	X'00000048'

ME* (Message descriptor extension structure identifier)

See the *MESID* field described in “MQMDE – Message descriptor extension” on page 104.

MESIDV	'MDEb'	

ME* (Message descriptor extension version)

See the *MEVER* field described in “MQMDE – Message descriptor extension” on page 104.

MEVER2	2	X'00000002'
MEVERC	2	X'00000002'

MEF* (Message descriptor extension flags)

See the *MEFLG* field described in “MQMDE – Message descriptor extension” on page 104.

MEFNON	0	X'00000000'

MS* (Message delivery sequence)

See the *MsgDeliverySequence* attribute described in “Attributes for local queues” on page 246.

MSPRIO	0	X'00000000'
MSFIFO	1	X'00000001'

MF* (Message flags)

See the *MDMFL* field described in “MQMD – Message descriptor” on page 59.

MFSEGI	0	X'00000000'
MFNONE	0	X'00000000'
MFSEGA	1	X'00000001'

MFSEG	2	X'00000002'
MFLSEG	4	X'00000004'
MFMIG	8	X'00000008'
MFLMIG	16	X'00000010'

MF* (Message-flags masks)

See Appendix C, “Report options” on page 379.

MFAUM	-1048576	X'FFF00000'
MFAUXM	1044480	X'000FF000'
MFRUM	4095	X'00000FFF'

MI* (Message identifier)

See the *MDMID* field described in “MQMD – Message descriptor” on page 59.

MINONE	X'00...00' (24 nulls)

MO* (Match options)

See the *GMMO* field described in “MQGMO – Get message options” on page 27.

MONONE	0	X'00000000'
MOMSGI	1	X'00000001'
MOCORI	2	X'00000002'
MOGRPI	4	X'00000004'
MOSEQN	8	X'00000008'
MOOFFS	16	X'00000010'

MT* (Message type)

See the *MDMT* field described in “MQMD – Message descriptor” on page 59.

MTSFST	1	X'00000001'
MTRQST	1	X'00000001'
MTRPLY	2	X'00000002'
MTRPRT	4	X'00000004'
MTDGRM	8	X'00000008'
MTSLST	65535	X'0000FFFF'
MTAFST	65536	X'00010000'
MTALST	999999999	X'3B9AC9FF'

OD* (Object descriptor length)

ODLENC	224	X'000000E0'

OD* (Object descriptor structure identifier)

See the *ODSID* field described in “MQOD – Object descriptor” on page 110.

ODSIDV	'ODbb'

OD* (Object descriptor version)

See the *ODVER* field described in "MQOD – Object descriptor" on page 110.

ODVER1	1	X'00000001'
ODVER2	2	X'00000002'
ODVERC	2	X'00000002'

OII* (Object instance identifier)

See the *RMOII* field described in "MQRMH –Reference message header" on page 140.

OIINON	X'00...00' (24 nulls)

OL* (Original length)

See the *MDOLN* field described in "MQMD – Message descriptor" on page 59.

OLUNDF	-1	X'FFFFFFFF'

OO* (Open options)

See the *OPTS* parameter described in "MQOPEN – Open object" on page 204.

OOINPQ	1	X'00000001'
OOINPS	2	X'00000002'
OOINPX	4	X'00000004'
OOBRW	8	X'00000008'
OOOUT	16	X'00000010'
OOINQ	32	X'00000020'
OOSET	64	X'00000040'
OOSAVA	128	X'00000080'
OOPASI	256	X'00000100'
OOPASA	512	X'00000200'
OOSETI	1024	X'00000400'
OOSETA	2048	X'00000800'
OOALTU	4096	X'00001000'
OOFIQ	8192	X'00002000'

OT* (Object type)

See the *ODOT* field described in "MQOD – Object descriptor" on page 110.

OTQ	1	X'00000001'
OTPRO	3	X'00000003'
OTQM	5	X'00000005'
OTCHAN	6	X'00000006'

PE* (Persistence)

See the *MDPER* field described in "MQMD – Message descriptor" on page 59, and the *DefPersistence* attribute described in "Attributes for all queues" on page 243.

PENPER	0	X'00000000'
PEPER	1	X'00000001'
PEQDEF	2	X'00000002'

PL* (Platform)

See the *Platform* attribute described in “Attributes for the queue-manager” on page 264.

PLMVS	1	X'00000001'
PLOS2	2	X'00000002'
PLAIX	3	X'00000003'
PLUNIX	3	X'00000003'
PL400	4	X'00000004'
PLWIN	5	X'00000005'
PLWINT	11	X'0000000B'

PM* (Put message options)

See the *PMOPT* field described in “MQPMO – Put message options” on page 120.

PMNONE	0	X'00000000'
PMSYP	2	X'00000002'
PMNSYP	4	X'00000004'
PMDEFC	32	X'00000020'
PMNMID	64	X'00000040'
PMNCID	128	X'00000080'
PMPASI	256	X'00000100'
PMPASA	512	X'00000200'
PMSETI	1024	X'00000400'
PMSETA	2048	X'00000800'
PMALTU	4096	X'00001000'
PMFIQ	8192	X'00002000'
PMNOC	16384	X'00004000'
PMLOGO	32768	X'00008000'

PM* (Put message options structure length)

PMLENC	176	X'000000B0'

PM* (Put message options structure identifier)

See the *PMSID* field described in “MQPMO – Put message options” on page 120.

PMSIDV	'PMOb'

PM* (Put message options version)

See the *PMVER* field described in “MQPMO – Put message options” on page 120.

PMVER1	1	X'00000001'
PMVER2	2	X'00000002'
PMVERC	2	X'00000002'

PF* (Put message record field flags)

See the *DHPRF* field described in “MQDH – Distribution header” on page 13.

PFNONE	0	X'00000000'
PFMID	1	X'00000001'
PFCID	2	X'00000002'

PFGID	4	X'00000004'
PFFB	8	X'00000008'
PFACC	16	X'00000010'

PR* (Priority)

See the *MDPRI* field described in "MQMD – Message descriptor" on page 59.

PRQDEF	-1	X'FFFFFFFF'

QA* (Inhibit get)

See the *InhibitGet* attribute described in "Attributes for all queues" on page 243.

QAGETA	0	X'00000000'
QAGETI	1	X'00000001'

QA* (Inhibit put)

See the *InhibitPut* attribute described in "Attributes for all queues" on page 243.

QAPUTA	0	X'00000000'
QAPUTI	1	X'00000001'

QA* (Backout hardening)

See the *HardenGetBackout* attribute described in "Attributes for local queues" on page 246.

QABNH	0	X'00000000'
QABH	1	X'00000001'

QA* (Queue shareability)

See the *Shareability* attribute described in "Attributes for local queues" on page 246.

QANSHR	0	X'00000000'
QASHR	1	X'00000001'

QD* (Queue definition type)

See the *DefinitionType* attribute described in "Attributes for local queues" on page 246.

QDPRE	1	X'00000001'
QDPERM	2	X'00000002'
QDTEMP	3	X'00000003'

QSIE* (Service interval events)

QSIENO	0	X'00000000'
QSIEHI	1	X'00000001'
QSIEOK	2	X'00000002'

QT* (Queue type)

See the *QType* attribute described in “Attributes for all queues” on page 243.

QTLOC	1	X'00000001'
QTMOD	2	X'00000002'
QTALS	3	X'00000003'
QTREM	6	X'00000006'

RC* (Reason code)

See Chapter 5, “Return codes” on page 275, and the *MDFB* field described in “MQMD – Message descriptor” on page 59. Note: the following list is in **numeric order**.

RCNONE	0	X'00000000'
RC2001	2001	X'000007D1'
RC2002	2002	X'000007D2'
RC2003	2003	X'000007D3'
RC2004	2004	X'000007D4'
RC2005	2005	X'000007D5'
RC2006	2006	X'000007D6'
RC2007	2007	X'000007D7'
RC2008	2008	X'000007D8'
RC2009	2009	X'000007D9'
RC2010	2010	X'000007DA'
RC2011	2011	X'000007DB'
RC2012	2012	X'000007DC'
RC2013	2013	X'000007DD'
RC2014	2014	X'000007DE'
RC2016	2016	X'000007E0'
RC2017	2017	X'000007E1'
RC2018	2018	X'000007E2'
RC2019	2019	X'000007E3'
RC2020	2020	X'000007E4'
RC2021	2021	X'000007E5'
RC2022	2022	X'000007E6'
RC2023	2023	X'000007E7'
RC2024	2024	X'000007E8'
RC2025	2025	X'000007E9'
RC2026	2026	X'000007EA'
RC2027	2027	X'000007EB'
RC2029	2029	X'000007ED'
RC2030	2030	X'000007EE'
RC2031	2031	X'000007EF'
RC2033	2033	X'000007F1'
RC2034	2034	X'000007F2'
RC2035	2035	X'000007F3'
RC2036	2036	X'000007F4'
RC2037	2037	X'000007F5'
RC2038	2038	X'000007F6'
RC2039	2039	X'000007F7'
RC2040	2040	X'000007F8'
RC2041	2041	X'000007F9'
RC2042	2042	X'000007FA'

RC2043	2043	X'000007FB'
RC2044	2044	X'000007FC'
RC2045	2045	X'000007FD'
RC2046	2046	X'000007FE'
RC2047	2047	X'000007FF'
RC2048	2048	X'00000800'
RC2049	2049	X'00000801'
RC2050	2050	X'00000802'
RC2051	2051	X'00000803'
RC2052	2052	X'00000804'
RC2053	2053	X'00000805'
RC2055	2055	X'00000807'
RC2056	2056	X'00000808'
RC2057	2057	X'00000809'
RC2058	2058	X'0000080A'
RC2059	2059	X'0000080B'
RC2061	2061	X'0000080D'
RC2062	2062	X'0000080E'
RC2063	2063	X'0000080F'
RC2065	2065	X'00000811'
RC2066	2066	X'00000812'
RC2067	2067	X'00000813'
RC2068	2068	X'00000814'
RC2069	2069	X'00000815'
RC2070	2070	X'00000816'
RC2071	2071	X'00000817'
RC2072	2072	X'00000818'
RC2075	2075	X'0000081B'
RC2076	2076	X'0000081C'
RC2077	2077	X'0000081D'
RC2078	2078	X'0000081E'
RC2079	2079	X'0000081F'
RC2080	2080	X'00000820'
RC2082	2082	X'00000822'
RC2085	2085	X'00000825'
RC2086	2086	X'00000826'
RC2087	2087	X'00000827'
RC2090	2090	X'0000082A'
RC2091	2091	X'0000082B'
RC2092	2092	X'0000082C'
RC2093	2093	X'0000082D'
RC2094	2094	X'0000082E'
RC2095	2095	X'0000082F'
RC2096	2096	X'00000830'
RC2097	2097	X'00000831'
RC2098	2098	X'00000832'
RC2099	2099	X'00000833'
RC2100	2100	X'00000834'
RC2101	2101	X'00000835'
RC2102	2102	X'00000836'
RC2103	2103	X'00000837'
RC2104	2104	X'00000838'
RC2105	2105	X'00000839'
RC2106	2106	X'0000083A'

RC2109	2109	X'0000083D'
RC2110	2110	X'0000083E'
RC2111	2111	X'0000083F'
RC2112	2112	X'00000840'
RC2113	2113	X'00000841'
RC2114	2114	X'00000842'
RC2115	2115	X'00000843'
RC2116	2116	X'00000844'
RC2117	2117	X'00000845'
RC2118	2118	X'00000846'
RC2119	2119	X'00000847'
RC2120	2120	X'00000848'
RC2151	2120	X'00000848'
RC2125	2125	X'0000084D'
RC2126	2126	X'0000084E'
RC2127	2127	X'0000084F'
RC2129	2129	X'00000851'
RC2130	2130	X'00000852'
RC2131	2131	X'00000853'
RC2132	2132	X'00000854'
RC2133	2133	X'00000855'
RC2135	2135	X'00000857'
RC2136	2136	X'00000858'
RC2137	2137	X'00000859'
RC2138	2138	X'0000085A'
RC2140	2140	X'0000085C'
RC2141	2141	X'0000085D'
RC2142	2142	X'0000085E'
RC2143	2143	X'0000085F'
RC2144	2144	X'00000860'
RC2145	2145	X'00000861'
RC2146	2146	X'00000862'
RC2148	2148	X'00000864'
RC2149	2149	X'00000865'
RC2150	2150	X'00000866'
RC2152	2152	X'00000868'
RC2153	2153	X'00000869'
RC2154	2154	X'0000086A'
RC2155	2155	X'0000086B'
RC2156	2156	X'0000086C'
RC2157	2157	X'0000086D'
RC2158	2158	X'0000086E'
RC2159	2159	X'0000086F'
RC2160	2160	X'00000870'
RC2161	2161	X'00000871'
RC2162	2162	X'00000872'
RC2163	2163	X'00000873'
RC2173	2173	X'0000087D'
RC2183	2183	X'00000887'
RC2184	2184	X'00000888'
RC2185	2185	X'00000889'
RC2186	2186	X'0000088A'
RC2187	2187	X'0000088B'
RC2191	2191	X'0000088F'

RC2192	2192	X'00000890'
RC2193	2193	X'00000891'
RC2194	2194	X'00000892'
RC2195	2195	X'00000893'
RC2196	2196	X'00000894'
RC2197	2197	X'00000895'
RC2198	2198	X'00000896'
RC2199	2199	X'00000897'
RC2201	2201	X'00000899'
RC2202	2202	X'0000089A'
RC2203	2203	X'0000089B'
RC2204	2204	X'0000089C'
RC2206	2206	X'0000089E'
RC2207	2207	X'0000089F'
RC2208	2208	X'000008A0'
RC2209	2209	X'000008A1'
RC2216	2216	X'000008A8'
RC2217	2217	X'000008A9'
RC2218	2218	X'000008AA'
RC2219	2219	X'000008AB'
RC2220	2220	X'000008AC'
RC2222	2222	X'000008AE'
RC2223	2223	X'000008AF'
RC2224	2224	X'000008B0'
RC2225	2225	X'000008B1'
RC2226	2226	X'000008B2'
RC2227	2227	X'000008B3'
RC2233	2233	X'000008B9'
RC2234	2234	X'000008BA'
RC2235	2235	X'000008BB'
RC2236	2236	X'000008BC'
RC2237	2237	X'000008BD'
RC2238	2238	X'000008BE'
RC2239	2239	X'000008BF'
RC2241	2241	X'000008C1'
RC2242	2242	X'000008C2'
RC2243	2243	X'000008C3'
RC2244	2244	X'000008C4'
RC2245	2245	X'000008C5'
RC2246	2246	X'000008C6'
RC2247	2247	X'000008C7'
RC2248	2248	X'000008C8'
RC2249	2249	X'000008C9'
RC2250	2250	X'000008CA'
RC2251	2251	X'000008CB'
RC2252	2252	X'000008CC'
RC2253	2253	X'000008CD'
RC2255	2255	X'000008CF'
RC2256	2256	X'000008D0'
RC2257	2257	X'000008D1'
RC2258	2258	X'000008D2'
RC2259	2259	X'000008D3'
RC2260	2260	X'000008D4'
RC2261	2261	X'000008D5'

RC2262	2262	X'000008D6'
RC2263	2263	X'000008D7'
RC2264	2264	X'000008D8'
RC2265	2265	X'000008D9'
RC2280	2280	X'000008E8'
RC2281	2281	X'000008E9'
RC2282	2282	X'000008EA'
RC2283	2283	X'000008EB'
RC2284	2284	X'000008EC'
RC2285	2285	X'000008ED'
RC2286	2286	X'000008EE'
RC2287	2287	X'000008EF'
RC2288	2288	X'000008F0'
RC2289	2289	X'000008F1'
RC2290	2290	X'000008F2'
RC2291	2291	X'000008F3'
RC2292	2292	X'000008F4'
RC2293	2293	X'000008F5'
RC2294	2294	X'000008F6'
RC2295	2295	X'000008F7'
RC2296	2296	X'000008F8'

RM* (Reference message header structure identifier)

See the *RMSID* field described in “MQRMH –Reference message header” on page 140.

RMSIDV	'RMHb'

RM* (Reference message header version)

See the *RMVER* field described in “MQRMH –Reference message header” on page 140.

RMVER1	1	X'00000001'
RMVERC	1	X'00000001'

RM* (Reference message header flags)

See the *RMFLG* field described in “MQRMH –Reference message header” on page 140.

RMNLST	0	X'00000000'
RMLAST	1	X'00000001'

RO* (Report options)

See the *MDREP* field described in “MQMD – Message descriptor” on page 59.

RONMI	0	X'00000000'
ROCMTC	0	X'00000000'
RODLQ	0	X'00000000'
RONONE	0	X'00000000'
ROPAN	1	X'00000001'
RONAN	2	X'00000002'
ROPCI	64	X'00000040'

ROPMI	128	X'00000080'
ROCOA	256	X'00000100'
ROCOAD	768	X'00000300'
ROCOAF	1792	X'00000700'
ROCOD	2048	X'00000800'
ROCODD	6144	X'00001800'
ROCODF	14336	X'00003800'
ROEXP	2097152	X'00200000'
ROEXPD	6291456	X'00600000'
ROEXPF	14680064	X'00E00000'
ROEXC	16777216	X'01000000'
ROEXCD	50331648	X'03000000'
ROEXCF	117440512	X'07000000'
RODISC	134217728	X'08000000'

RO* (Report-options masks)

See Appendix C, "Report options" on page 379.

RORUM	270270464	X'101C0000'
ROAUM	-270532353	X'EFE000FF'
ROAUXM	261888	X'0003FF00'

SEG* (Segmentation)

See the *GMSEG* field described in "MQGMO – Get message options" on page 27.

SEGIHB	'b'
SEGALW	'A'

SP* (Syncpoint)

See the *SyncPoint* attribute described in "Attributes for the queue-manager" on page 264.

SPNAVL	0	X'00000000'
SPAVL	1	X'00000001'

SS* (Segment status)

See the *GMSST* field described in "MQGMO – Get message options" on page 27.

SSNSEG	'b'
SSLSEG	'L'
SSSEG	'S'

TC* (Trigger control)

See the *TriggerControl* attribute described in "Attributes for local queues" on page 246.

TCOFF	0	X'00000000'
TCON	1	X'00000001'

TM* (Trigger message structure identifier)

See the *TMSID* field described in “MQTM – Trigger message” on page 151.

TMSIDV	'TMbb'	

TM* (Trigger message version)

See the *TMVER* field described in “MQTM – Trigger message” on page 151.

TMVER1	1	X'00000001'
TMVERC	1	X'00000001'

TC* (Trigger message character format structure identifier)

See the *TCSID* field described in “MQTMC – Trigger message (character format)” on page 156.

TCSIDV	'TMCb'	

TC* (Trigger message character format version)

See the *TCVER* field described in “MQTMC – Trigger message (character format)” on page 156.

TCVER1	'bbb1'	
TCVER2	'bbb2'	
TCVERC	'bbb2'	

TT* (Trigger type)

See the *TriggerType* attribute described in “Attributes for local queues” on page 246.

TTNONE	0	X'00000000'
TTFRST	1	X'00000001'
TTEVRY	2	X'00000002'
TTDPTH	3	X'00000003'

US* (Usage)

See the *Usage* attribute described in “Attributes for local queues” on page 246.

USNORM	0	X'00000000'
USTRAN	1	X'00000001'

WI* (Wait interval)

See the *GMWI* field described in “MQGMO – Get message options” on page 27.

WIULIM	-1	X'FFFFFFFF'

XR* (Data-conversion-exit response)

See the *DXRES* field described in “MQDXP - Data conversion header” on page 391.

XROK	0	X'00000000'
XRFAIL	1	X'00000001'

XQ* (Transmission queue header structure identifier)

See the *XQSID* field described in “MQXQH –Transmission queue header” on page 159.

XQSIDV	'XQHb'

XQ* (Transmission queue header version)

See the *XQVER* field described in “MQXQH –Transmission queue header” on page 159.

XQVER1	1	X'00000001'
XQVERC	1	X'00000001'

Part 2. Building your application

Chapter 7. Building your application

The OS/400 publications describe how to build executable applications from the programs you write. This chapter describes the additional tasks, and the changes to the standard tasks, you must perform when building MQSeries for AS/400 applications to run under OS/400.

In addition to coding the MQI calls in your source code, you must add the appropriate language statements to include the MQSeries for AS/400 copy files for the RPG language. You should make yourself familiar with the contents of these files—their names, and a brief description of their contents are given in the following section .

MQSeries copy files

MQSeries for AS/400 provides copy files to assist you with writing your applications in the RPG programming language. They are suitable for use with the IBM ILE RPG/400 Compiler (5716-RG1).

The copy files that MQSeries for AS/400 provides to assist with the writing of channel exits are described in the *MQSeries Intercommunication* book.

The names of the MQSeries for AS/400 copy files for RPG have the prefix CMQ. They have a suffix of G. There are separate copy files containing the named constants, and one file for each of the structures. The copy files are listed in Table 3 on page 9.

Note: For ILE RPG/400 they are supplied as members of file QRPGLESRC in library QMQM.

The structure declarations do not contain **DS** statements. This allows the application to declare a data structure (or a multiple-occurrence data structure) by coding the **DS** statement and using the **/COPY** statement to copy in the remainder of the declaration:

For ILE RPG/400 the statement is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7
D* Declare an MQMD data structure
D MQMD        DS
D/COPY CMQMDG
```

Preparing your programs to run

To create an executable MQSeries for AS/400 application, you have to compile the source code you have written.

To do this for ILE RPG/400, you can use the usual OS/400 commands, CRTRPGMOD and CRTPGM.

After creating your *MODULE, you also need to reference the service program QMQM/AMQZSTUB when creating your ILE RPG/400 program using the CRTPGM command.

© Copyright IBM Corp. 1994, 1998

Make sure that the library containing the copy files (QMQM) is in the library list when you perform the compilation. QMQM *must* also be in the library list when you run the application.

Syncpoints in MQSeries for AS/400 applications

To start the MQSeries for AS/400 commitment control facilities, use the STRCMTCTL command.

Note: The default value of *Commitment definition scope* is *ACTGRP. This must be defined as *JOB for MQSeries for AS/400.

All MQSeries for AS/400 code runs in a single, named activation group: QMQM.

If you call MQPUT, MQPUT1 or MQGET, specifying MQPMO_SYNCPOINT or MQGMO_SYNCPOINT, when MQSeries for AS/400 is not registered as an API commitment resource inside your commitment definition, MQSeries for AS/400 adds itself to the definition. This is typically the first such call in a job. While there are any API commitment resources registered under a particular commitment definition, you cannot end commitment control for that definition.

MQSeries for AS/400 normally removes its registration, as an API commitment resource, when you disconnect from the queue manager. The precise mechanism used depends on how you connected to the queue manager:

- If you connected to the queue manager explicitly, using the MQCONN call, you must issue an MQDISC call to disconnect from the queue manager.
- If you connected to the queue manager implicitly, by opening an object using the MQOPEN call without a previous MQCONN call, you must use the MQCLOSE call for each of the opened objects, to disconnect from the queue manager.

If you try to disconnect from the queue manager while there are pending MQPUT, MQPUT1 or MQGET operations in the current unit of work, MQSeries for AS/400 remains registered as an API commitment resource so that it can be notified of the next commit or backout. When the next syncpoint is reached, MQSeries for AS/400 commits or backs out the changes as required. However, MQSeries for AS/400 is then unable to remove itself from the commitment definition, which remains active until the job ends.

If you attempt to issue an ENDCMTCTL command for that commitment definition, message CPF8355 is issued, indicating that pending changes were active. This message also appears in the job log when the job ends. In order to avoid this situation, ensure that you commit or backout all pending MQSeries for AS/400 operations before you disconnect from the queue manager. This will enable you to end commitment control.

To commit or rollback (backout) your unit of work, use one of the programming languages that supports the commitment control

CL commands	– COMMIT and ROLLBACK
ILE C Programming Functions	– _Rcommit and _Rrollback
OPM RPG	– COMIT and ROLBK
ILE RPG	– COMMIT and ROLBK

Syncpoints in CICS for AS/400 applications

MQSeries for AS/400 participates in CICS for AS/400 units of work. You can use the MQI within a CICS for AS/400 application to put and get messages inside the current unit of work.

You can use the EXEC CICS SYNCPOINT command to establish a syncpoint that includes the MQSeries for AS/400 operations. To back out all changes up to the previous syncpoint, you can use the EXEC CICS SYNCPOINT ROLLBACK command.

If you use MQPUT, MQPUT1, or MQGET with the MQPMO_SYNCPOINT, or MQGMO_SYNCPOINT, option set in a CICS for AS/400 application, you cannot log off CICS for AS/400 until MQSeries for AS/400 has removed its registration as an API commitment resource. Therefore, you should commit or back out any pending put or get operations before you disconnect from the queue manager. This will allow you to log off CICS for OS/400.

Part 3. MQSeries sample applications

© Copyright IBM Corp. 1994, 1998

Chapter 8. Sample programs

This chapter describes the sample programs delivered with MQSeries for AS/400 for RPG. The samples demonstrate typical uses of the Message Queue Interface (MQI).

The samples are not intended to demonstrate general programming techniques, so some error checking that you may want to include in a production program has been omitted. However, these samples are suitable for use as a base for your own message queuing programs.

The source code for all the samples is provided with the product; this source includes comments that explain the message queuing techniques demonstrated in the programs.

There are three sets of sample programs:

1. **OPM RPG programs**

   The source exists in QMQMSAMP/QRPGSRC. The members are named `AMQ1xxx4`, where `xxx` indicates the sample function. Copy members exist in QMQM/QRPGSRC.

2. **ILE RPG programs using the MQI through a call to QMQM**

   The source exists in QMQMSAMP/QRPGLESRC. The members are named `AMQ2xxx4`, where `xxx` indicates the sample function. Copy members exist in QMQM/QRPGLESRC. Each member name has a suffix of “R”.

3. **ILE RPG programs using prototyped calls to the MQI**

   The source exists in QMQMSAMP/QRPGLESRC. The members are named `AMQ3xxx4`, where `xxx` indicates the sample function. Copy members exist in QMQM/QRPGLESRC. Each member name has a suffix of “G”.

   Note: Sample trigger programs only exist for this set.

Table 45 gives a complete list of the sample programs delivered with MQSeries for AS/400 V3R1 or later, and shows the names of the programs in each of the supported programming languages. Notice that their names all start with the prefix AMQ, the fourth character in the name indicates the programming language.

Note: This chapter tells you how to use the ILE RPG/400 compiler. with prototyped calls to the MQI.

Table 45 (Page 1 of 2). Names of the sample programs

	RPG (ILE)
Put samples	AMQ3PUT4
Browse samples	AMQ3GBR4
Get samples	AMQ3GET4
Request samples	AMQ3REQ4
Echo samples	AMQ3ECH4
Inquire samples	AMQ3INQ4

© Copyright IBM Corp. 1994, 1998

Table 45 (Page 2 of 2). Names of the sample programs	
	RPG (ILE)
Set samples	AMQ3SET4
Trigger Monitor sample	AMQ3TRG4
Trigger Server sample	AMQ3SRV4

In addition to these, the MQSeries for AS/400sample option includes a sample data file, AMQSDATA, which can be used as input to certain sample programs. and sample CL programs that demonstrate administration tasks. The CL samples are described in the *MQSeries for AS/400 Administration Guide*. You could use the sample CL program AMQSAMP4 to create queues to use with the sample programs described in this chapter .

For information on how to run the sample programs, see “Preparing and running the sample programs” on page 357.

Features demonstrated in the sample programs

Table 46 on page 357 shows the techniques demonstrated by the MQSeries for AS/400 sample programs. Some techniques occur in more than one sample program, but only one program is listed in the table. All the samples open and close queues using the MQOPEN and MQCLOSE calls, so these techniques are not listed separately in the table.

Table 46. Sample programs demonstrating use of the MQI

Technique	RPG (ILE)
Using the MQCONN and MQDISC calls	AMQ3ECH4 or AMQ3INQ4
Implicitly connecting and disconnecting	AMQ3PUT4
Putting messages using the MQPUT call	AMQ3PUT4
Putting a single message using the MQPUT1 call	AMQ3ECH4 or AMQ3INQ4
Replying to a request message	AMQ3INQ4
Getting messages (no wait)	AMQ3GBR4
Getting messages (wait with a time limit)	AMQ3GET4
Getting messages (with data conversion)	AMQ3ECH4
Browsing a queue	AMQ3GBR4
Using a shared input queue	AMQ3INQ4
Using an exclusive input queue	AMQ3REQ4
Using the MQINQ call	AMQ3INQ4
Using the MQSET call	AMQ3SET4
Using a reply-to queue	AMQ3REQ4
Requesting exception messages	AMQ3REQ4
Accepting a truncated message	AMQ3GBR4
Using a resolved queue name	AMQ3GBR4
Trigger processing	AMQ3SRV4 or AMQ3TRG4

Note: All the sample programs produce a spool file that contains the results of the processing.

Preparing and running the sample programs

Before you can run the MQSeries for AS/400 sample programs, you must compile them as you would any other MQSeries for AS/400 applications. To do this, you can use the OS/400 commands CRTRPGMOD and CRTPGM.

When you create the `AMQ3xxx4` programs, you need to specify BNDSRVPGM(QMQM/AMQZSTUB) in the CRTPGM command. This includes the various MQ procedures in your program.

The sample programs are provided in library QMQMSAMP as members of QRPGLESRC. They use the copy files provided in library QMQM, so make sure this library is in the library list when you compile them. The RPG compiler gives information messages because the samples do not use many of the variables that are declared in the copy files.

Running the sample programs

You can use your own queues when you run the samples, or you can run AMQSAMP4 to create some sample queues. The source for this program is shipped in file QCLSRC in library QMQMSAMP. It can be compiled using the CRTCLPGM command.

To call one of the sample programs, use a command like:

```
CALL PGM(QMQMSAMP/AMQ3PUT4) PARM('Queue Name')
```

where `Queue Name` *must* be 48 characters in length, which you achieve by padding the queue name with the required number of blanks.

Note that for the Inquire and Set sample programs, the sample definitions created by AMQSAMP4 cause the C versions of these samples to be triggered. If you want to trigger the RPG versions, you must change the process definitions SYSTEM.SAMPLE.ECHOPROCESS and SYSTEM.SAMPLE.INQPROCESS and SYSTEM.SAMPLE.SETPROCESS. You can use the CHGMQMPRC command (described in the *MQSeries for AS/400 Administration Guide* to do this, or edit and run AMQSAMP4 with the alternative definition.

The Put sample program

The Put sample program, AMQ3PUT4, puts messages on a queue using the MQPUT call.

To start the program, call the program and give the name of your target queue as a program parameter. The program puts a set of fixed messages on the queue; these messages are taken from the data block at the end of the program source code. A sample put program is AMQ3PUT4 in library QMQMSAMP.

Using this example program, the command is:

```
CALL PGM(QMQMSAMP/AMQ3PUT4) PARM('Queue Name')
```

where `Queue Name` *must* be 48 characters in length, which you achieve by padding the queue name with the required number of blanks.

Design of the Put sample program

The program uses the MQOPEN call with the OOOUT option to open the target queue for putting messages. The results are output to a spool file. If it cannot open the queue, the program writes an error message containing the reason code returned by the MQOPEN call. To keep the program simple, on this and on subsequent MQI calls, the program uses default values for many of the options.

For each line of data contained in the source code, the program reads the text into a buffer and uses the MQPUT call to create a datagram message containing the text of that line. The program continues until either it reaches the end of the input or the MQPUT call fails. If the program reaches the end of the input, it closes the queue using the MQCLOSE call.

The Browse sample program

The Browse sample program, AMQ3GBR4, browses messages on a queue using the MQGET call.

The program retrieves copies of all the messages on the queue you specify when you call the program; the messages remain on the queue. You could use the supplied queue SYSTEM.SAMPLE.LOCAL; run the Put sample program first to put some messages on the queue. You could use the queue SYSTEM.SAMPLE.ALIAS, which is an alias name for the same local queue. The program continues until it reaches the end of the queue or an MQI call fails.

An example of a command to call the RPG program is:

```
CALL PGM(QMQMSAMP/AMQ3GBR4) PARM('Queue Name')
```

where `Queue Name` *must* be 48 characters in length, which you achieve by padding the queue name with the required number of blanks. Therefore, if you are using SYSTEM.SAMPLE.LOCAL as your target queue, you will need 29 blank characters.

Design of the Browse sample program

The program opens the target queue using the MQOPEN call with the OOBRW option. If it cannot open the queue, the program writes an error message to its spool file, containing the reason code returned by the MQOPEN call.

For each message on the queue, the program uses the MQGET call to copy the message from the queue, then displays the data contained in the message. The MQGET call uses these options:

GMBRWN
: After the MQOPEN call, the browse cursor is positioned logically before the first message in the queue, so this option causes the *first* message to be returned when the call is first made.

GMNWT
: The program does not wait if there are no messages on the queue.

GMATM
: The MQGET call specifies a buffer of fixed size. If a message is longer than this buffer, the program displays the truncated message, together with a warning that the message has been truncated.

The program demonstrates how you must clear the *MDMID* and *MDCID* fields of the MQMD structure after each MQGET call because the call sets these fields to the values contained in the message it retrieves. Clearing these fields means that successive MQGET calls retrieve messages in the order in which the messages are held in the queue.

The program continues to the end of the queue; at this point the MQGET call returns the RC2033 (no message available) reason code and the program displays a warning message. If the MQGET call fails, the program writes an error message that contains the reason code in its spool file.

The program then closes the queue using the MQCLOSE call.

The Get sample program

The Get sample program, AMQ3GET4, gets messages from a queue using the MQGET call.

When the program is called, it removes messages from the specified queue. You could use the supplied queue SYSTEM.SAMPLE.LOCAL; run the Put sample program first to put some messages on the queue. You could use the SYSTEM.SAMPLE.ALIAS queue, which is an alias name for the same local queue. The program continues until the queue is empty or an MQI call fails.

An example of a command to call the RPG program is:

```
CALL PGM(QMQMSAMP/AMQ3GET4) PARM('Queue Name')
```

where `Queue Name` *must* be 48 characters in length, which you achieve by padding the queue name with the required number of blanks. Therefore, if you are using SYSTEM.SAMPLE.LOCAL as your target queue, you will need 29 blank characters.

Design of the Get sample program

The program opens the target queue for getting messages; it uses the MQOPEN call with the OOINPQ option. If it cannot open the queue, the program writes an error message containing the reason code returned by the MQOPEN call in its spool file.

For each message on the queue, the program uses the MQGET call to remove the message from the queue; it then displays the data contained in the message. The MQGET call uses the GMWT option, specifying a wait interval (*GMWI*) of 15 seconds, so that the program waits for this period if there is no message on the queue. If no message arrives before this interval expires, the call fails and returns the RC2033 (no message available) reason code.

The program demonstrates how you must clear the *MDMID* and *MDCID* fields of the MQMD structure after each MQGET call because the call sets these fields to the values contained in the message it retrieves. Clearing these fields means that successive MQGET calls retrieve messages in the order in which the messages are held in the queue.

The MQGET call specifies a buffer of fixed size. If a message is longer than this buffer, the call fails and the program stops.

The program continues until either the MQGET call returns the RC2033 (no message available) reason code or the MQGET call fails. If the call fails, the program displays an error message that contains the reason code.

The program then closes the queue using the MQCLOSE call.

The Request sample program

The Request sample program, AMQ3REQ4, demonstrates client/server processing. The sample is the client that puts request messages on a queue that is processed by a server program. It waits for the server program to put a reply message on a reply-to queue.

The Request sample puts a series of request messages on a queue using the MQPUT call. These messages specify SYSTEM.SAMPLE.REPLY as the reply-to queue. The program waits for reply messages, then displays them. Replies are sent only if the target queue (which we will call the *server queue*) is being processed by a server application, or if an application is triggered for that purpose (the Inquire and Set sample programs are designed to be triggered). The sample waits 5 minutes for the first reply to arrive (to allow time for a server application to be triggered) and 15 seconds for subsequent replies, but it can end without getting any replies.

To start the program, call the program and give the name of your target queue as a program parameter. The program puts a set of fixed messages on the queue; these messages are taken from the data block at the end of the program source code.

Using triggering with the Request sample

To run the sample using triggering, start the trigger server program, AMQ3SRV4, against the required initiation queue in one job, then start AMQ3REQ4 in another job. This means that the trigger server is ready when the Request sample program sends a message.

Notes:

1. The samples use the SYSTEM SAMPLE TRIGGER queue as the initiation queue for SYSTEM.SAMPLE.ECHO, SYSTEM.SAMPLE.INQ, or SYSTEM.SAMPLE.SET local queues. Alternatively, you can define your own initiation queue..
2. The sample definitions created by AMQSAMP4 cause the C version of the sample to be triggered. If you want to trigger the RPG version, you must change the process definitions SYSTEM.SAMPLE.ECHOPROCESS and SYSTEM.SAMPLE.INQPROCESS and SYSTEM.SAMPLE.SETPROCESS. You can use the CHGMQMPRC command (described in the *MQSeries for AS/400 Administration Guide* to do this, or edit and run your own version of AMQSAMP4.
3. You need to compile the trigger server program from the source provided in QMQMSAMP/QRPGLESRC.

Depending on the trigger process you want to run, AMQ3REQ4 should be called with the parameter specifying request messages to be placed on one of these sample server queues:

- SYSTEM.SAMPLE.ECHO (for the Echo sample programs)
- SYSTEM.SAMPLE.INQ (for the Inquire sample programs)
- SYSTEM.SAMPLE.SET (for the Set sample programs)

A flow chart for the SYSTEM.SAMPLE.ECHO program is shown in Figure 1 on page 364. Using the example the command to issue the RPG program request to this server is:

```
CALL PGM(QMQMSAMP/AMQ3REQ4) PARM('SYSTEM.SAMPLE.ECHO
+ 30 blank characters')
```

because the queue name *must* be 48 characters in length.

Note: This sample queue has a trigger type of FIRST, so if there are already messages on the queue before you run the Request sample, server applications are not triggered by the messages you send.

If you want to attempt further examples, you can try the following variations:

- Use AMQ3TRG4 instead of AMQ3SRV4 to submit the job instead, but potential job submission delays could make it less easy to follow what is happening.
- Use the SYSTEM.SAMPLE.INQ and SYSTEM.SAMPLE.SET sample queues. Using the example data file the commands to issue the RPG program requests to these servers are, respectively:

  ```
  CALL PGM(QMQMSAMP/AMQ3INQ4) PARM('SYSTEM.SAMPLE.INQ
  + 31 blank characters')
  CALL PGM(QMQMSAMP/AMQ3SET4) PARM('SYSTEM.SAMPLE.SET
  + 31 blank characters')
  ```

  because the queue name *must* be 48 characters in length.

  These sample queues also have a trigger type of FIRST.

Design of the Request sample program

The program opens the server queue so that it can put messages. It uses the MQOPEN call with the OOOUT option. If it cannot open the queue, the program displays an error message containing the reason code returned by the MQOPEN call.

The program then opens the reply-to queue called SYSTEM.SAMPLE.REPLY so that it can get reply messages. For this, the program uses the MQOPEN call with the OOINPX option. If it cannot open the queue, the program displays an error message containing the reason code returned by the MQOPEN call.

For each line of input, the program then reads the text into a buffer and uses the MQPUT call to create a request message containing the text of that line. On this call the program uses the ROEXCD report option to request that any report messages sent about the request message will include the first 100 bytes of the message data. The program continues until either it reaches the end of the input or the MQPUT call fails.

The program then uses the MQGET call to remove reply messages from the queue, and displays the data contained in the replies. The MQGET call uses the GMWT option, specifying a wait interval (*GMWI*) of 5 minutes for the first reply (to allow time for a server application to be triggered) and 15 seconds for subsequent replies. The program waits for these periods if there is no message on the queue. If no message arrives before this interval expires, the call fails and returns the RC2033 (no message available) reason code. The call also uses the GMATM option, so messages longer than the declared buffer size are truncated.

The program demonstrates how you must clear the *MDMID* and *MDCOD* fields of the MQMD structure after each MQGET call because the call sets these fields to the values contained in the message it retrieves. Clearing these fields means that successive MQGET calls retrieve messages in the order in which the messages are held in the queue.

The program continues until either the MQGET call returns the RC2033 (no message available) reason code or the MQGET call fails. If the call fails, the program displays an error message that contains the reason code.

The program then closes both the server queue and the reply-to queue using the MQCLOSE call. 47 shows the changes to the Echo sample program that are necessary to run the Inquire and Set sample programs.

Note: The details for the Echo sample program are included as a reference.

Table 47. Client/Server sample program details

Program name	SYSTEM/SAMPLE queue	Program started
Echo	ECHO	AMQ3ECH4
Inquire	INQ	AMQ3INQ4
Set	SET	AMQ3SET4

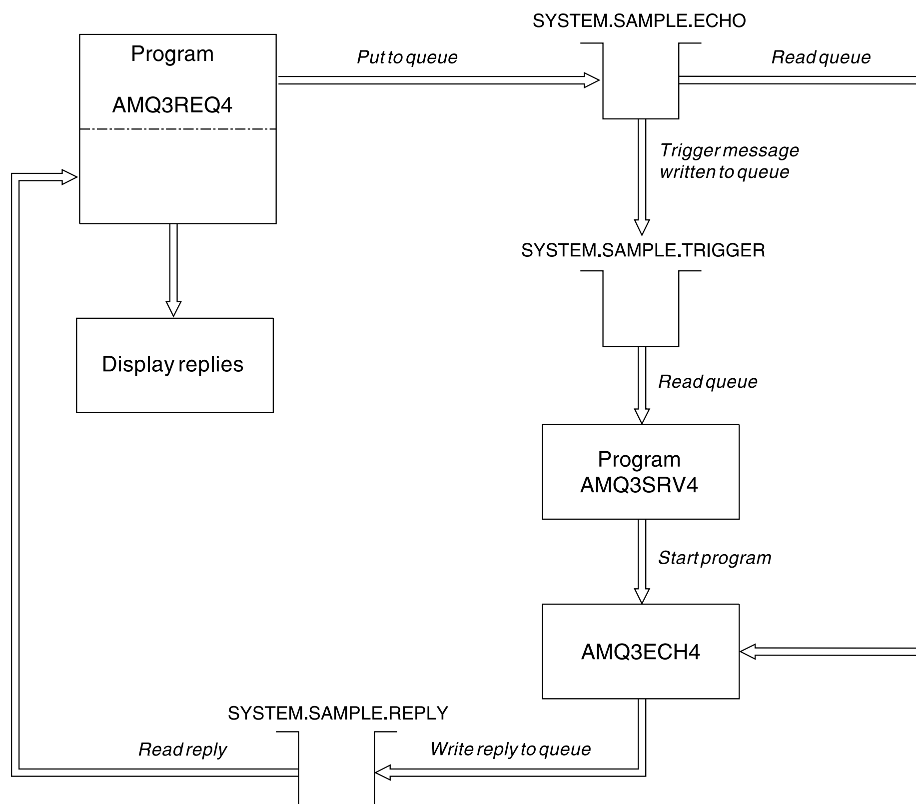


Figure 1. Sample Client/Server (Echo) program flowchart

The Echo sample program

The Echo sample programs return the message send to a reply queue. The program is named AMQ3ECH4

The programs are intended to run as triggered programs, so their only input is the data read from the queue named in the trigger message structure.

For the triggering process to work, you must ensure that the Echo sample program you want to use is triggered by messages arriving on queue SYSTEM.SAMPLE.ECHO. To do this, specify the name of the Echo sample program you want to use in the *ApplId* field of the process definition SYSTEM.SAMPLE.ECHOPROCESS. (For this, you can use the CHGMQMPRC command, described in the *MQSeries for AS/400 Administration Guide*.) The sample queue has a trigger type of FIRST, so if there are already messages on the queue before you run the Request sample, the Echo sample is not triggered by the messages you send.

When you have set the definition correctly, first start AMQ3SRV4 in one job, then start AMQ3REQ4 in another. You could use AMQ3TRG4 instead of AMQ3SRV4, but potential job submission delays could make it less easy to follow what is happening.

Use the Request sample programs to send messages to queue SYSTEM.SAMPLE.ECHO. The Echo sample programs send a reply message

containing the data in the request message to the reply-to queue specified in the request message.

Design of the Echo sample program

When the program is triggered, it explicitly connects to the default queue manager using the MQCONN call. Although this is not necessary for MQSeries for AS/400, this means you could use the same program on other platforms without changing the source code.

The program then opens the queue named in the trigger message structure it was passed when it started. (For clarity, we will call this the *request queue*.) The program uses the MQOPEN call to open this queue for shared input.

The program uses the MQGET call to remove messages from this queue. This call uses the GMATM and GMWT options, with a wait interval of 5 seconds. The program tests the descriptor of each message to see if it is a request message; if it is not, the program discards the message and displays a warning message.

For each request message removed from the request queue, the program uses the MQPUT call to put a reply message on the reply-to queue. This message contains the contents of the request message.

When there are no messages remaining on the request queue, the program closes that queue and disconnects from the queue manager.

This program can also respond to messages sent to the queue from platforms other than MQSeries for AS/400, although no sample is supplied for this situation. To make the ECHO program work, you:

- Write a program, correctly specifying the *Format*, *Encoding*, and *CCSID* fields, to send text request messages.

  The ECHO program requests the queue manager to perform message data conversion, if this is needed.
- Specify CONVERT(*YES) on the MQSeries for AS/400 sending channel, if the program you have written does not provide similar conversion for the reply.

The Inquire sample program

The Inquire sample program, AMQ3INQ4, inquires about some of the attributes of a queue using the MQINQ call.

The program is intended to run as a triggered program, so its only input is an MQTMC (trigger message) structure that contains the name of a target queue whose attributes are to be inquired.

For the triggering process to work, you must ensure that the Inquire sample program is triggered by messages arriving on queue SYSTEM.SAMPLE.INQ. To do this, specify the name of the Inquire sample program in the *ApplId* field of the SYSTEM.SAMPLE.INQPROCESS process definition. (For this, you can use the CHGMQMPRC command, described in the *MQSeries for AS/400 Administration Guide*.) The sample queue has a trigger type of FIRST, so if there are already messages on the queue before you run the Request sample, the Inquire sample is not triggered by the messages you send.

When you have set the definition correctly, first start AMQ3SRV4 in one job, then start AMQ3REQ4 in another. You could use AMQ3TRG4 instead of AMQ3SRV4, but potential job submission delays could make it less easy to follow what is happening.

Use the Request sample program to send request messages, each containing just a queue name, to queue SYSTEM.SAMPLE.INQ. For each request message, the Inquire sample program sends a reply message containing information about the queue specified in the request message. The replies are sent to the reply-to queue specified in the request message.

Design of the Inquire sample program

When the program is triggered, it explicitly connects to the default queue manager using the MQCONN call. Although this is not necessary for MQSeries for AS/400, this means you could use the same program on other platforms without changing the source code.

The program then opens the queue named in the trigger message structure it was passed when it started. (For clarity, we will call this the *request queue*.) The program uses the MQOPEN call to open this queue for shared input.

The program uses the MQGET call to remove messages from this queue. This call uses the GMATM and GMWT options, with a wait interval of 5 seconds. The program tests the descriptor of each message to see if it is a request message; if it is not, the program discards the message and displays a warning message.

For each request message removed from the request queue, the program reads the name of the queue (which we will call the *target queue*) contained in the data and opens that queue using the MQOPEN call with the OOINQ option. The program then uses the MQINQ call to inquire about the values of the *InhibitGet*, *CurrentQDepth*, and *OpenInputCount* attributes of the target queue.

If the MQINQ call is successful, the program uses the MQPUT call to put a reply message on the reply-to queue. This message contains the values of the 3 attributes.

If the MQOPEN or MQINQ call is unsuccessful, the program uses the MQPUT call to put a *report* message on the reply-to queue. In the *MDFB* field of the message descriptor of this report message is the reason code returned by either the MQOPEN or MQINQ call, depending on which one failed.

After the MQINQ call, the program closes the target queue using the MQCLOSE call.

When there are no messages remaining on the request queue, the program closes that queue and disconnects from the queue manager.

The Set sample program

The Set sample program, AMQ3SET4, inhibits put operations on a queue by using the MQSET call to change the queue's *InhibitPut* attribute.

The program is intended to run as a triggered program, so its only input is an MQTMC (trigger message) structure that contains the name of a target queue whose attributes are to be inquired.

For the triggering process to work, you must ensure that the Set sample program is triggered by messages arriving on queue SYSTEM.SAMPLE.SET. To do this, specify the name of the Set sample program in the *ApplId* field of the process definition SYSTEM.SAMPLE.SETPROCESS. (For this, you can use the CHGMQMPRC command, described in the *MQSeries for AS/400 Administration Guide*.) The sample queue has a trigger type of FIRST, so if there are already messages on the queue before you run the Request sample, the Set sample is not triggered by the messages you send.

When you have set the definition correctly, first start AMQ3SRV4 in one job, then start AMQ3REQ4 in another. You could use AMQ3TRG4 instead of AMQ3SRV4, but potential job submission delays could make it less easy to follow what is happening.

Use the Request sample program to send request messages, each containing just a queue name, to queue SYSTEM.SAMPLE.SET. For each request message, the Set sample program sends a reply message containing a confirmation that put operations have been inhibited on the specified queue. The replies are sent to the reply-to queue specified in the request message.

Design of the Set sample program

When the program is triggered, it explicitly connects to the default queue manager using the MQCONN call. Although this is not necessary for MQSeries for AS/400, this means you could use the same program on other platforms without changing the source code.

The program then opens the queue named in the trigger message structure it was passed when it started. (For clarity, we will call this the *request queue*.) The program uses the MQOPEN call to open this queue for shared input.

The program uses the MQGET call to remove messages from this queue. This call uses the GMATM and GMWT options, with a wait interval of 5 seconds. The program tests the descriptor of each message to see if it is a request message; if it is not, the program discards the message and displays a warning message.

For each request message removed from the request queue, the program reads the name of the queue (which we will call the *target queue*) contained in the data and opens that queue using the MQOPEN call with the OOSET option. The program then uses the MQSET call to set the value of the *InhibitPut* attribute of the target queue to QAPUTI.

If the MQSET call is successful, the program uses the MQPUT call to put a reply message on the reply-to queue. This message contains the string `PUT inhibited`.

If the MQOPEN or MQSET call is unsuccessful, the program uses the MQPUT call to put a *report* message on the reply-to queue. In the `MDFB` field of the message descriptor of this report message is the reason code returned by either the MQOPEN or MQSET call, depending on which one failed.

After the MQSET call, the program closes the target queue using the MQCLOSE call.

When there are no messages remaining on the request queue, the program closes that queue and disconnects from the queue manager.

The Triggering sample programs

MQSeries for AS/400 supplies two Triggering sample programs that are written in ILE/RPG. The programs are:

AMQ3TRG4 This is a trigger monitor for the OS/400 environment. It submits an OS/400 job for the application to be started, but this means there is a processing overhead associated with each trigger message.

AMQ3SRV4 This is a trigger server for the OS/400 environment. For each trigger message, this server runs the start command in its own job to start the specified application. The trigger server can call CICS transactions.

C language versions of these samples are also available as executable programs in library QMQM, called AMQSTRG4 and AMQSERV4.

The AMQ3TRG4 sample trigger monitor

AMQ3TRG4 is a trigger monitor. It takes one parameter: the name of the initiation queue it is to serve. AMQSAMP4 defines a sample initiation queue, SYSTEM.SAMPLE.TRIGGER, that you can use when you try the sample programs.

AMQ3TRG4 submits an OS/400 job for each valid trigger message it gets from the initiation queue.

Design of the trigger monitor

The trigger monitor opens the initiation queue and gets messages from the queue, specifying an unlimited wait interval.

The trigger monitor submits an OS/400 job to start the application specified in the trigger message, and passes an MQTMC (a character version of the trigger message) structure. The environment data in the trigger message is used as job submission parameters.

Finally, the program closes the initiation queue.

The AMQ3SRV4 sample trigger server

AMQ3SRV4 is a trigger server. It takes one parameter: the name of the initiation queue it is to serve. AMQSAMP4 defines a sample initiation queue, SYSTEM.SAMPLE.TRIGGER, that you can use when you try the sample programs.

For each trigger message, AMQ3SRV4 runs a start command in its own job to start the specified application.

Using the example trigger queue the command to issue is:

```
CALL PGM(QMQM/AMQ3SRV4) PARM('Queue Name')
```

where `Queue Name` *must* be 48 characters in length, which you achieve by padding the queue name with the required number of blanks. Therefore, if you are using SYSTEM.SAMPLE.TRIGGER as your target queue, you will need 28 blank characters.

Design of the trigger server

The design of the trigger server is similar to that of the trigger monitor, except the trigger server:

- Allows CICS as well as OS/400 applications
- Does not use the environment data from the trigger message
- Calls OS/400 applications in its own job (or uses STRCICSUSR to start CICS applications) rather than submitting an OS/400 job
- Opens the initiation queue for shared input, so many trigger servers can run at the same time

Note: Programs started by AMQ3SRV4must not use the MQDISC call because this will stop the trigger server. If programs started by AMQ3SRV4use the MQCONN call, they will get the RC2002 reason code.

Ending the Triggering sample programs

A trigger monitor program can be ended by the sysrequest option 2 (ENDRQS) or by inhibiting gets from the trigger queue. If the sample trigger queue is used the command is:

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') GETENBL(*NO)
```

Note: To start triggering again on this queue, you *must* enter the command:

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') GETENBL(*YES)
```

Running the samples using remote queues

You can demonstrate remote queuing by running the samples on connected message queue managers.

Program AMQSAMP4 provides a local definition of a remote queue (SYSTEM.SAMPLE.REMOTE) that uses a remote queue manager named `OTHER`. To use this sample definition, change `OTHER` to the name of the second message queue manager you want to use. You must also set up a message channel between your two message queue managers; for information on how to do this, see the *MQSeries Intercommunication* manual.

The Request sample program puts its own local queue manager name in the *MDRM* field of messages it sends. The Inquire and Set samples send reply messages to the queue and message queue manager named in the *MDRQ* and *MDRM* fields of the request messages they process.

Part 4. Appendixes

Appendix A. Rules for validating MQI options

This appendix explains the situations that produce an RC2046 reason code from an MQOPEN, MQPUT, MQPUT1, MQGET, or MQCLOSE call.

MQOPEN

For the options of the MQOPEN call:

- Only valid options are allowed.
- At least *one* of the following must be specified:
  - OOINPX
  - OOINPS
  - OOINPQ
  - OOBRW
  - OOOUT
  - OOINQ
  - OOSET
- Only *one* of the following is allowed:
  - OOINPX
  - OOINPS
  - OOINPQ
- If OOSAVA is specified, one of the OOINP* options must also be specified.
- If one of the OOSET* or OOPAS* options is specified, OOOUT must also be specified.

MQPUT

For the put-message options:

- Only valid options are allowed.
- The combination of PMSYP and PMNSYP is not allowed.
- Only *one* of the following is allowed:
  - PMNOC
  - PMDEFC
  - PMPASI
  - PMPASA
  - PMSETI
  - PMSETA
- PMALTU is not allowed (it is valid only on the MQPUT1 call).

© Copyright IBM Corp. 1994, 1998

MQPUT1

For the put-message options, the rules are the same as for the MQPUT call, except that PMALTU is allowed.

MQGET

For the get-message options:

- Only valid options are allowed.
- Only *one* of the following is allowed:
  - GMSYP
  - GMPSYP
  - GMNSYP
- Only *one* of the following is allowed:
  - GMBRWF
  - GMBRWN
  - GMBRWC
  - GMMUC
- GMSYP is not allowed with any of the following:

  GMBRWF
  GMBRWN
  GMBRWC
  GMLK
  GMUNLK
- GMPSYP is not allowed with any of the following:

  GMBRWF
  GMBRWN
  GMBRWC
  GMCMPM
  GMLK
  GMUNLK
- If GMLK is specified, one of the following must also be specified:

  GMBRWF
  GMBRWN
  GMBRWC
- If GMUNLK is specified, only the following are allowed:

  GMNWT
  GMNSYP

MQCLOSE

For the options of the MQCLOSE call:

- Only valid options are allowed.
- The combination of CODEL and COPURG is not allowed.

Appendix B. Machine encodings

This appendix describes the structure of the *MDENC* field in the message descriptor MQMD (see page 77).

The *MDENC* 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:

ENIMSK
: Mask for binary-integer encoding.

  This subfield occupies bit positions 28 through 31 within the *MDENC* field.

ENDMSK
: Mask for packed-decimal-integer encoding.

  This subfield occupies bit positions 24 through 27 within the *MDENC* field.

ENFMSK
: Mask for floating-point encoding.

  This subfield occupies bit positions 20 through 23 within the *MDENC* field.

ENRMSK
: Mask for reserved bits.

  This subfield occupies bit positions 0 through 19 within the *MDENC* field.

Binary-integer encoding

The following values are valid for the binary-integer encoding:

ENIUND
: Undefined integer encoding.

  Binary integers are represented using an encoding that is undefined.

ENINOR
: 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
  - 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

© Copyright IBM Corp. 1994, 1998

ENIREV
: Reversed integer encoding.

  Binary integers are represented in the same way as ENINOR, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as ENINOR.

Packed-decimal-integer encoding

The following values are valid for the packed-decimal-integer encoding:

ENDUND
: Undefined packed-decimal encoding.

  Packed-decimal integers are represented using an encoding that is undefined.

ENDNOR
: 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.

ENDREV
: Reversed packed-decimal encoding.

  Packed-decimal integers are represented in the same way as ENDNOR, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as ENDNOR.

Floating-point encoding

The following values are valid for the floating-point encoding:

ENFUND
: Undefined floating-point encoding.

  Floating-point numbers are represented using an encoding that is undefined.

ENFNOR
: Normal IEEE float encoding.

Floating-point numbers are represented using the standard IEEE[2]

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.

ENFREV
: Reversed IEEE float encoding.

Floating-point numbers are represented in the same way as ENFNOR, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as ENFNOR.

ENF390
: 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 *MDENC* field in MQMD, the relevant constants that describe the required encodings should be added together. Be sure to combine only one of the ENI* encodings with one of the END* encodings and one of the ENF* encodings.

Analyzing encodings

The *MDENC* field contains subfields; because of this, applications that need to examine the integer, packed decimal, or float encoding should use the technique described below.

Using arithmetic

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 *MDENC* field by A; call the result B.
3. Divide B by 16; call the result C.

[2] The Institute of Electrical and Electronics Engineers

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 48.

Table 48. 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 C. Report options

This appendix concerns the *MDREP* and *MDMFL* fields that are part of the message descriptor MQMD specified on the MQGET, MQPUT, and MQPUT1 calls (see page 61). 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 *MDREP* 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:

RORUM
: Mask for unsupported report options that are rejected.

  This mask identifies the bit positions within the *MDREP* 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 CCFAIL and reason code RC2061.

  This subfield occupies bit positions 3, and 11 through 13.

ROAUM
: Mask for unsupported report options that are accepted.

  This mask identifies the bit positions within the *MDREP* 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 CCWARN with reason code RC2104 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:

  ROCMTC
  RODLQ
  RODISC
  ROEXC
  ROEXCD
  ROEXCF

© Copyright IBM Corp. 1994, 1998

ROEXP
ROEXPD
ROEXPF
RONAN
RONMI
RONONE
ROPAN
ROPCI
ROPMI

ROAUXM
Mask for unsupported report options that are accepted only in certain circumstances.

This mask identifies the bit positions within the *MDREP* 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 *ODMN* and *ODON* fields in the object descriptor specified on the MQOPEN or MQPUT1 call is not a local transmission queue).

Completion code CCWARN with reason code RC2104 are returned if these conditions are satisfied, and CCFAIL with reason code RC2061 if not.

This subfield occupies bit positions 14 through 23.

The following report options are included in this subfield:

ROCOA
ROCOAD
ROCOAF
ROCOD
ROCODD
ROCODF

If there are any options specified in the *MDREP* 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 *MDREP* 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 CCWARN 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 *MDREP* field contains subfields; because of this, applications that need to check whether the sender of the message requested a particular report should use the technique described below.

Using arithmetic

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	ROCOA
COD	ROCOD
Exception	ROEXC
Expiration	ROEXP

   Call the value A.

2. Divide the *MDREP* 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 ROEXC, test E for equality with each of the following to determine what was specified by the sender of the message:

   RONONE
   ROEXC
   ROEXCD
   ROEXCF

   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 ROPMI or ROPCI 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 *MDMFL* 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 *MDMFL* 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:

MFRUM
: Mask for unsupported message flags that are rejected.

  This mask identifies the bit positions within the *MDMFL* 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 CCFAIL and reason code RC2249.

  This subfield occupies bit positions 20 through 31.

  The following message flags are included in this subfield:

  MFLMIG
  MFLSEG
  MFMIG
  MFSEG
  MFSEGA

MFAUM
: Mask for unsupported message flags that are accepted.

  This mask identifies the bit positions within the *MDMFL* 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 CCOK.

  This subfield occupies bit positions 0 through 11.

MFAUXM
: Mask for unsupported message flags that are accepted only in certain circumstances.

  This mask identifies the bit positions within the *MDMFL* 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 *ODMN* and *ODON* fields in the

object descriptor specified on the MQOPEN or MQPUT1 call is not a local transmission queue).

Completion code CCOK is returned if these conditions are satisfied, and CCFAIL with reason code RC2249 if not.

This subfield occupies bit positions 12 through 19.

If there are any flags specified in the *MDMFL* 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 *MDMFL* 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.

Appendix D. 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 GMCONV option to be specified on the MQGET call.

The following are described:

- The processing performed by the queue manager in response to the GMCONV 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 386.
- Special considerations for the conversion of report messages; see "Conversion of report messages" on page 390.
- The parameters passed to the data-conversion exit; see MQDATACONVEXIT on page 398.
- A call that can be used from the exit in order to convert character data between different representations; see MQXCNVC on page 403.
- The data-structure parameter which is specific to the exit; see "MQDXP - Data conversion header" on page 391.

Conversion processing

The queue manager performs the following actions if the GMCONV 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 *MDCSI* and *MDENC* values in the control information in the message are identical to those in the *MSGDSC* parameter.
   - The length of the application message data is zero.
   - The length of the *BUFFER* parameter is zero.

   In these cases the message is returned without conversion to the application issuing the MQGET call; the *MDCSI* and *MDENC* values in the *MSGDSC* 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
CCOK	RCNONE
CCWARN	RC2079
CCWARN	RC2080

© Copyright IBM Corp. 1994, 1998

The following steps are performed only if the *MDCSI* or *MDENC* value in the control information in the message differs from that in the *MSGDSC* parameter, and there is data to be converted:

2. If the *MDFMT* field in the control information in the message has the value FMNONE, the message is returned unconverted, with completion code CCWARN and reason code RC2110.

   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 GMATM option was *not* specified, the message is returned unconverted, with completion code CCWARN and reason code RC2080.
   - If the GMATM option *was* specified, the completion code is set to CCWARN, the reason code is set to RC2079, and conversion processing continues.

5. If the message can be accommodated in the buffer without truncation, or the GMATM 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 CCWARN and reason code RC2110.

   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 will usually be one of the following combinations:

Completion code	Reason code
CCOK	RCNONE
CCWARN	RC2079

   If the conversion fails (for whatever reason), the queue manager returns the unconverted message to the application, with the *MDCSI* and *MDENC* fields in the *MSGDSC* parameter set to the values in the control information in the message, and with completion code CCWARN. 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:

FMADMN
FMCMD1

FMCMD2
FMDLH
FMEVNT
FMIMS
FMIMVS
FMMDE
FMPCF
FMRMH
FMSTR
FMTM
FMXQH

1. If the message expands during conversion, and exceeds the size of the *BUFFER* parameter, the following is done:
   - If the GMATM option was *not* specified, the message is returned unconverted, with completion code CCWARN and reason code RC2120.
   - If the GMATM option *was* specified, the message is truncated, the completion code is set to CCWARN, the reason code is set to RC2079, 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 *DATLEN* parameter is the length of the *unconverted* message before truncation.
5. The data returned to the application is never partially converted; either all of the data returned is converted, or none of it is. For example, if the integers in the data can be converted, but the character strings cannot (because the character-set identifier is not recognized), none of the data is converted.
6. If the *MDCSI* or *MDENC* fields in the control information of the message being retrieved, or in the *MSGDSC* 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 *MDENC* 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 CCWARN and one of the RC2111, RC2112, RC2113, RC2114 or RC2115, RC2116, RC2117, RC2118 reason codes (as appropriate); the *MDCSI* and *MDENC* fields in the *MSGDSC* 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 *MDCSI* and *MDENC* fields in the *MSGDSC* parameter are those specified by the application issuing the MQGET call.

7. In all cases, if the message is returned to the application unconverted the completion code is set to CCWARN, and the *MDCSI* and *MDENC* fields in the *MSGDSC* parameter are set to the values appropriate to the unconverted data. This is done for FMNONE 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 *MDCSI* and *MDENC* fields in the *MSGDSC* parameter.)

   When an error is diagnosed, either a specific reason code is returned, or the general reason code RC2119. The reason code returned depends on the diagnostic capabilities of the underlying data-conversion service.

8. If completion code CCWARN is returned, and more than one reason code is relevant, the order of precedence is as follows:

   a. The following reason takes precedence over all others:

      RC2079

   b. Next in precedence is the following reason:

      RC2110

   c. The order of precedence within this final group is not defined:

      RC2120
      RC2119
      RC2111
      RC2113
      RC2114
      RC2112
      RC2115
      RC2117
      RC2118
      RC2116

9. On completion of the MQGET call:

   - The following reason code indicates that the message was converted successfully:

     RCNONE

   - The following reason code indicates that the message *may* have been converted successfully (check the *MDCSI* and *MDENC* fields in the *MSGDSC* parameter to find out):

     RC2079

- 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:

10. With the exception of the format FMSTR, none of the built-in formats can be converted from or to double-byte character sets (DBCS); only single-byte character sets (SBCS) can be used with these formats.

    If DBCS character sets are specified with these formats, the message is returned unconverted, with completion code CCWARN and reason code RC2111 or RC2115, as appropriate.

11. 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 FMADMN message, care must be taken to ensure that the application does not attempt to access data beyond the end of the data returned.

12. If the format name is FMDLH, 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 *DLFMT*, *DLCSI*, and *DLENC* 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 *DLCSI* and *DLENC* 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 *DLFMT* field in the MQDLH structure, or performs the conversion itself (if *DLFMT* is the name of a built-in format).

    If the MQGET call returns a completion code of CCWARN, 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 *MDCSI* and *MDENC* fields in the *MSGDSC* parameter, and those in the MQDLH structure, in order to determine which of the above applies.

13. If the format name is FMXQH, 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 *MDFMT*, *MDCSI*, and *MDENC* fields in the MQMD structure contained *within* the MQXQH. For each subsequent MQ header structure present, the *MDFMT*, *MDCSI*, and *MDENC* 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 GMCONV option is specified for an FMXQH 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 *MDFMT*, *MDCSI*, and *MDENC* fields in the *MSGDSC* 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 GMCONV option specified must reset the *MDCSI* and *MDENC* fields in the *MSGDSC* parameter to the values desired for the application message data, prior to each MQGET call.

- The values of the *MDFMT*, *MDCSI*, and *MDENC* 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 *MSGDSC* 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 GMCONV 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 RO*D 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 RO*F, or specifies RO*D 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 *MDFMT* 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 GMCONV 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 RCNONE (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 CCOK and reason code RCNONE should be returned by the exit. The length of the message data to be converted is passed to the exit as the *INLEN* parameter.

For information on data conversion, see the *MQSeries Application Programming Guide*.

MQDXP - Data conversion header

The following table guides you to the appropriate page for each field.

Table 49. Fields in MQDXP

This field...	Describes...	See page ...
DXSID	Structure identifier	392
DXVER	Structure version number	392
DXAOP	Application options	393
DXENC	Encoding required by application	393
DXCSI	Character set required by application	393
DXLEN	Length in bytes of message data	393
DXCC	Completion code	394
DXREA	Reason code qualifying *DXCC*	394
DXRES	Response from exit	395
DXHCN	Connection handle	396

The MQDXP structure is a parameter that is passed to the data-conversion exit. See the description of the MQDATACONVEXIT call for details of the data conversion exit.

Only the *DXLEN*, *DXCC*, *DXREA* and *DXRES* fields in MQDXP may be changed by the exit; changes to other fields are ignored. However, the *DXLEN* 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 XRFAIL in *DXRES*; however, the queue manager ignores the values of the *DXCC* and *DXREA* 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:

- *DXRES* field not XROK and not XRFAIL
- *DXCC* field not CCOK and not CCWARN
- *DXLEN* field less than zero, or *DXLEN* field changed when the message being converted is a segment that contains only part of a logical message.

Fields

DXSID (4-byte character string)
Structure identifier.

The value must be:

DXSIDV
Identifier for data conversion exit parameter structure.

This is an input field to the exit.

DXVER (10-digit signed integer)
Structure version number.

The value must be:

DXVER1
Version number for data-conversion exit parameter structure.

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

DXVERC
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 *DXVER* 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.

DXXOP (10-digit signed integer)
Reserved.

This is a reserved field; its value is 0.

DXAOP (10-digit signed integer)
Application options.

This is a copy of the *GMOPT* field of the MQGMO structure specified by the application issuing the MQGET call. The exit may need to examine these to ascertain whether the GMATM option was specified.

This is an input field to the exit.

DXENC (10-digit signed integer)
Encoding required by application.

This is the encoding required by the application issuing the MQGET call; see the *MDENC* field in the MQMD structure for more details.

If the conversion is successful, the exit should copy this to the *MDENC* field in the message descriptor.

This is an input field to the exit.

DXCSI (10-digit signed integer)
Coded character-set identifier required by application.

This is the coded character-set identifier of the character set required by the application issuing the MQGET call; see the *MDCSI* field in the MQMD structure for more details. If the application specifies the special value CSQM 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 *MDCSI* field in the message descriptor.

This is an input field to the exit.

DXLEN (10-digit signed integer)
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 *DXLEN*. The size of the message actually provided to the exit is always given by the *INLEN* parameter of the exit, irrespective of any truncation that may have occurred.

Truncation is indicated by the *DXREA* field having the value RC2079 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 *DATLEN* 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 *DXLEN* is always greater then zero.

This is an input/output field to the exit.

DXCC (10-digit signed integer)
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 CCWARN, 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 *CMPCOD* parameter of the MQGET call; only CCOK and CCWARN are valid. See the description of the *DXREA* field for recommendations on how the exit should set this field on output.

This is an input/output field to the exit.

DXREA (10-digit signed integer)
Reason code qualifying *DXCC*.

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 RC2079, indicating that the message was truncated in order fit into the buffer provided by the application, and RC2119, 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 *DXREA* had the value RC2079 on input to the exit, the *DXREA* and *DXCC* fields should not be altered, irrespective of whether the conversion succeeds or fails.

  (If the *DXCC* field is not CCOK, the application which retrieves the message can identify a conversion failure by comparing the returned *MDENC* and *MDCSI* 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, RC2079 should be returned in preference to any of the reasons that indicate conversion failure.)

- If *DXREA* had any other value on input to the exit:

  – If the conversion succeeds, *DXCC* should be set to CCOK and *DXREA* set to RCNONE.

  – If the conversion fails, or the message expands and has to be truncated to fit in the buffer, *DXCC* should be set to CCWARN (or

left unchanged), and *DXREA* 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 GMATM option:

- If it did specify that option, reason RC2079 should be returned.
- If it did not specify that option, the message should be returned unconverted, with reason code RC2120.

The reason codes listed below are recommended for use by the exit to indicate the reason that conversion failed, but the exit can set other values from the set of RC* codes if deemed appropriate.

Note: If the message cannot be converted successfully, the exit *must* return XRFAIL in the *DXRES* field, in order to cause the queue manager to return the unconverted message. This is true regardless of the reason code returned in the *DXREA* field.

RC2120
: (2120, X'848') Converted message too big for application buffer.

RC2119
: (2119, X'847') Application message data not converted.

RC2111
: (2111, X'83F') Source coded character set identifier not valid.

RC2113
: (2113, X'841') Packed-decimal encoding in message not recognized.

RC2114
: (2114, X'842') Floating-point encoding in message not recognized.

RC2112
: (2112, X'840') Source integer encoding not recognized.

RC2115
: (2115, X'843') Target coded character set identifier not valid.

RC2117
: (2117, X'845') Packed-decimal encoding specified by receiver not recognized.

RC2118
: (2118, X'846') Floating-point encoding specified by receiver not recognized.

RC2116
: (2116, X'844') Target integer encoding not recognized.

RC2079
: (2079, X'81F') Truncated message returned (processing completed).

This is an input/output field to the exit.

DXRES (10-digit signed integer)
: 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:

XROK
Conversion was successful.

If the exit specifies this value, the queue manager returns the following to the application which issued the MQGET call:

- The value of the *DXCC* field on output from the exit
- The value of the *DXREA* field on output from the exit
- The value of the *DXLEN* field on output from the exit
- The contents of the exit's output buffer *OUTBUF*; the number of bytes returned is the lesser of the exit's *OUTLEN* parameter, and the value of the *DXLEN* field on output from the exit
- The value of the *MDENC* field in the exit's message descriptor parameter on output from the exit
- The value of the *MDCSI* field in the exit's message descriptor parameter on output from the exit

XRFAIL
Conversion was unsuccessful.

If the exit specifies this value, the queue manager returns the following to the application which issued the MQGET call:

- The value of the *DXCC* field on output from the exit
- The value of the *DXREA* field on output from the exit
- The value of the *DXLEN* field on *input* to the exit
- The contents of the exit's input buffer *INBUF*; the number of bytes returned is given by the *INLEN* parameter

If the exit has altered *INBUF*, the results are undefined.

DXRES is an output field from the exit.

DXHCN (10-digit signed integer)
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.

RPG declaration (ILE)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQDXP Structure
D*
D* Structure identifier
D  DXSID                  1      4
D* Structure version number
D  DXVER                  5      8I 0
D* Reserved
D  DXXOP                  9     12I 0
D* Application options
D  DXAOP                 13     16I 0
D* Encoding required by application
D  DXENC                 17     20I 0
D* Coded character-set identifier required by application
```

```
D  DXCSI                 21     24I 0
D* Length in bytes of message data
D  DXLEN                 25     28I 0
D* Completion code
D  DXCC                  29     32I 0
D* Reason code qualifying CompCode
D  DXREA                 33     36I 0
D* Response from exit
D  DXRES                 37     40I 0
D* Connection handle
D  DXHCN                 41     44I 0
```

RPG declaration (OPM)

```
I*..1....:....2....:....3....:....4....:....5....:....6....:....7..
I* MQDXP Structure
I*
I* Structure identifier
I                                        1   4 DXSID
I* Structure version number
I                                    B   5   80DXVER
I* Reserved
I                                    B   9  120DXXOP
I* Application options
I                                    B  13  160DXAOP
I* Encoding required by application
I                                    B  17  200DXENC
I* Coded character-set identifier required by application
I                                    B  21  240DXCSI
I* Length in bytes of message data
I                                    B  25  280DXLEN
I* Completion code
I                                    B  29  320DXCC
I* Reason code qualifying CompCode
I                                    B  33  360DXREA
I* Response from exit
I                                    B  37  400DXRES
I* Connection handle
I                                    B  41  440DXHCN
```

MQDATACONVEXIT - Data conversion exit

This call definition describes the parameters that are passed to the data-conversion exit. No entry point called MQDATACONVEXIT is actually provided by the queue manager (see usage note 11 on page 401).

This definition is part of the MQSeries Data Conversion Interface (DCI), which is one of the MQSeries framework interfaces.

```
exitname(MQDXP, MQMD, INLEN, INBUF, OUTLEN, OUTBUF)
```

Parameters

MQDXP (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 header" on page 391 for details of the fields in this structure.

MQMD (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 *MDFMT*, *MDENC*, and *MDCSI* of the unconverted message contained in *INBUF*.

  Note: The *MQMD* 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 *MDVER* field in *MQMD* to verify that the fields that the exit needs to access are present in the structure.

  On OS/400, the exit is passed a version-2 MQMD.

  On output, the exit should change the *MDENC* and *MDCSI* 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.

INLEN (10-digit signed integer) - input
: Length in bytes of *INBUF*.

  This is the length of the input buffer *INBUF*, and specifies the number of bytes to be processed by the exit. *INLEN* 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.

INBUF (1-byte bit string×*INLEN*) - 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 *INBUF*; if this parameter is altered, the results are undefined.

OUTLEN (10-digit signed integer) - input
Length in bytes of *OUTBUF*.

This is the length of the output buffer *OUTBUF*, 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.

OUTBUF (1-byte bit string×*OUTLEN*) - output
Buffer containing the converted message.

On output from the exit, if the conversion was successful (as indicated by the value XROK in the *DXRES* field of the *MQDXP* parameter), *OUTBUF* 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.

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 GMCONV option is specified on the MQGET call
   - The *MDFMT* field in the message descriptor is not FMNONE
   - The message is not already in the required representation; that is, one or both of the message's *MDCSI* and *MDENC* 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 RCNONE or RC2079

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 GMATM option on the MQGET call.

In this case, the *DXREA* field in the *MQDXP* parameter on input to the exit will have the value RC2079.

- 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 390 for more details).

  In this case, the *DXREA* field in the *MQDXP* parameter on input to the exit will have the value RCNONE (if the receiving application provided a buffer that was big enough for the message).

Thus the value of the *DXREA* 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 *INLEN* parameter will be *less than* the length implied by the format name contained in the *MDFMT* field in the message descriptor. The exit should therefore check the value of *INLEN* 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 *INLEN* is less than the value expected, the exit should return XRFAIL in the *DXRES* field of the *MQDXP* parameter, with the *DXCC* and *DXREA* fields set to CCWARN and RC2110 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 *INBUF*. If the conversion completes successfully, the exit should leave the *DXREA* field in the *MQDXP* parameter unchanged. This has the effect of returning RC2079 if the message was truncated by the receiver’s queue manager, and RCNONE 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 *OUTBUF*. In this case the exit must decide whether to truncate the message; the *DXAOP* field in the *MQDXP* parameter will indicate whether the receiving application specified the GMATM option.

4. Generally it is recommended that all of the data in the message provided to the exit in *INBUF* 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 *OUTBUF* 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 environment-specific 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 *MDFMT* since the application 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 *MDCSI* or *MDENC* 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. *MDFMT* 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 RC2219, or other unpredictable errors.
11. No entry point called MQDATACONVEXIT is actually provided by the queue manager. The name of the exit should be the same as the format name (the name contained in the *MDFMT* field in MQMD), although this is not required in all environments.

RPG invocation (ILE)

```
C*..1...:....2...:....3...:....4...:....5...:....6...:....7..
C                     CALLP     exitname(MQDXP : MQMD : INLEN :
C                                        INBUF : OUTLEN : OUTBUF)
```

The prototype definition for the call is:

```
D*..1...:....2...:....3...:....4...:....5...:....6...:....7..
Dexitname        PR                  EXTPROC('exitname')
D* Data-conversion exit parameter block
D MQDXP                         44A
D* Message descriptor
D MQMD                         364A
D* Length in bytes of INBUF
D INLEN                         10I 0 VALUE
D* Buffer containing the unconverted message
D INBUF                           *   VALUE
D* Length in bytes of OUTBUF
D OUTLEN                        10I 0 VALUE
D* Buffer containing the converted message
D OUTBUF                          *   VALUE
```

RPG invocation (OPM)

```
C*..1...:....2...:....3...:....4...:....5...:....6...:....7..
C                     CALL 'exitname'
C* Data-conversion exit parameter block
C                     PARM           MQDXP
C* Message descriptor
C                     PARM           MQMD
C* Length in bytes of INBUF
C                     PARM           INLEN   90
C* Buffer containing the unconverted message
C                     PARM           INBUF   n
C* Length in bytes of OUTBUF
C                     PARM           OUTLEN  90
C* Buffer containing the converted message
C                     PARM           OUTBUF  n
```

Declare the structure parameters as follows:

```
I*..1...:....2...:....3...:....4...:....5...:....6...:....7..
I* Data-conversion exit parameter block
IMQDXP       DS
I/COPY CMQDXPR
I* Message descriptor
IMQMD        DS
I/COPY CMQMDR
```

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.

```
MQXCNVC (HCONN, OPTS, SRCCSI, SRCLEN, SRCBUF, TGTCSI, TGTLEN, TGTBUF,
         DATLEN, CMPCOD, REASON)
```

Parameters

HCONN (10-digit signed integer) - 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 *DXHCN* field of the MQDXP structure; this handle is not necessarily the same as the handle specified by the application which issued the MQGET call.

On OS/400, the following special value can be specified for *HCONN*:

HCDEFH
: Default connection handle.

OPTS (10-digit signed integer) - 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).

Default-conversion option: The following option controls the use of default character conversion:

DCCDEF
: 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 this option 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.

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, UCS2 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 DCCS* values should be specified, combined with one of the DCCT*values:

DCCSNA
: Source encoding is the default for the environment and programming language.

DCCSNO
: Source encoding is normal.

DCCSRE
: Source encoding is reversed.

DCCSUN
: Source encoding is undefined.

DCCTNA
: Target encoding is the default for the environment and programming language.

DCCTNO
: Target encoding is normal.

DCCTRE
: Target encoding is reversed.

DCCTUN
: Target encoding is undefined.

The encoding values defined above can be added directly to the *OPTS* field. However, if the source or target encoding is obtained from the *MDENC* field in the MQMD or other structure, the following processing must be done:

1. The integer encoding must be extracted from the *MDENC* field by eliminating the float and packed-decimal encodings; see "Analyzing encodings" on page 377 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 *OPTS* field. These factors are:

   DCCSFA
   : Factor for source encoding

   DCCTFA
   : 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 (DCC*UN). 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 UCS2 character set), the call fails with reason code RC2112 or RC2116 as appropriate.

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

DCCNON
: No options specified.

  DCCNON 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.

SRCCSI (10-digit signed integer) - input
: Coded character set identifier of string before conversion.

  This is the coded character set identifier of the input string in *SRCBUF*.

SRCLEN (10-digit signed integer) - input
: Length of string before conversion.

  This is the length in bytes of the input string in *SRCBUF*; it must be zero or greater.

SRCBUF (1-byte character string×*SRCLEN*) - input
: String to be converted.

  This is the buffer containing the string to be converted from one character set to another.

TGTCSI (10-digit signed integer) - input
: Coded character set identifier of string after conversion.

  This is the coded character set identifier of the character set to which *SRCBUF* is to be converted.

TGTLEN (10-digit signed integer) - input
: Length of output buffer.

  This is the length in bytes of the output buffer *TGTBUF*; it must be zero or greater.

TGTBUF (1-byte character string×*TGTLEN*) - output
: String after conversion.

  This is the string after it has been converted to the character set defined by *TGTCSI*. The converted string can be shorter or longer than the unconverted string.

If *TGTBUF* is too small to accommodate the converted string, the string is truncated to fit and the call completes with CCWARN and reason code RC2120. The string is truncated in a way that ensures it remains a valid SBCS, DBCS, or mixed SBCS/DBCS string; this may result in the number of valid bytes returned in *TGTBUF* being less than *TGTLEN*. The *DATLEN* parameter indicates the number of valid bytes returned.

DATLEN (10-digit signed integer) - output
Length of output string.

This is the length of the string returned in the output buffer *TGTBUF*. The converted string can be shorter or longer than the unconverted string.

CMPCOD (10-digit signed integer) - output
Completion code.

It is one of the following:

CCOK
Successful completion.
CCWARN
Warning (partial completion).
CCFAIL
Call failed.

REASON (10-digit signed integer) - output
Reason code qualifying *CMPCOD*.

If *CMPCOD* is CCOK:

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

If *CMPCOD* is CCWARN:

RC2120
(2120, X'848') Converted message too big for application buffer.

If *CMPCOD* is CCFAIL:

RC2010
(2010, X'7DA') Data length parameter not valid.
RC2150
(2150, X'866') DBCS string not valid.
RC2018
(2018, X'7E2') Connection handle not valid.
RC2046
(2046, X'7FE') Options not valid or not consistent.
RC2102
(2102, X'836') Insufficient system resources available.
RC2145
(2145, X'861') Source buffer parameter not valid.
RC2111
(2111, X'83F') Source coded character set identifier not valid.
RC2112
(2112, X'840') Source integer encoding not recognized.
RC2143
(2143, X'85F') Source length parameter not valid.

RC2071
(2071, X'817') Insufficient storage available.
RC2146
(2146, X'862') Target buffer parameter not valid.
RC2115
(2115, X'843') Target coded character set identifier not valid.
RC2116
(2116, X'844') Target integer encoding not recognized.
RC2144
(2144, X'860') Target length parameter not valid.
RC2195
(2195, X'893') Unexpected error occurred.

For more information on these reason codes, see Chapter 5, “Return codes” on page 275.

RPG invocation (ILE)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                   CALLP     MQXCNVC(HCONN : OPTS : SRCCSI :
C                                       SRCLEN : SRCBUF :
C                                       TGTCSI : TGTLEN :
C                                       TGTBUF : DATLEN :
C                                       CMPCOD : REASON)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
DMQXCNVC        PR                  EXTPROC('MQXCNVC')
D* Connection handle
D HCONN                         10I 0 VALUE
D* Options that control the action of MQXCNVC
D OPTS                          10I 0 VALUE
D* Coded character set identifier of string before conversion
D SRCCSI                        10I 0 VALUE
D* Length of string before conversion
D SRCLEN                        10I 0 VALUE
D* String to be converted
D SRCBUF                          *   VALUE
D* Coded character set identifier of string after conversion
D TGTCSI                        10I 0 VALUE
D* Length of output buffer
D TGTLEN                        10I 0 VALUE
D* String after conversion
D TGTBUF                          *   VALUE
D* Length of output string
D DATLEN                        10I 0
D* Completion code
D CMPCOD                        10I 0
D* Reason code qualifying CMPCOD
D REASON                        10I 0
```

RPG invocation (OPM)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                   CALL 'MQXCNVC'
C* Connection handle
C                   PARM           HCONN   90
C* Options that control the action of MQXCNVC
C                   PARM           OPTS    90
C* Coded character set identifier of string before conversion
C                   PARM           SRCCSI  90
C* Length of string before conversion
C                   PARM           SRCLEN  90
C* String to be converted
C                   PARM           SRCBUF  n
C* Coded character set identifier of string after conversion
C                   PARM           TGTCSI  90
C* Length of output buffer
C                   PARM           TGTLEN  90
C* String after conversion
C                   PARM           TGTBUF  n
C* Length of output string
C                   PARM           DATLEN  90
C* Completion code
```

```
C                      PARM           CMPCOD  90
C* Reason code qualifying CMPCOD
C                      PARM           REASON  90
```

Appendix E. Notices

The following paragraph does not apply to any 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 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.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. 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. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

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 Laboratory Counsel, MP151, IBM United Kingdom Laboratories, 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.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, U.S.A.

This publication contains sample programs. Permission is hereby granted to copy and store the sample programs into a data processing machine and to use the stored copies for internal study and instruction only. No permission is granted to use the sample programs for any other purpose.

Programming interface information

This book is intended to help you to write application programs that run under MQSeries for AS/400. This book documents General-use Programming Interface and Associated Guidance Information provided by MQSeries for AS/400.

General-use programming interfaces allow the customer to write programs that obtain the services of MQSeries for AS/400.

© Copyright IBM Corp. 1994, 1998

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both.

AFP	AIX	AS/400
AT	BookManager	CICS
FFST	First Failure Support Technology	IBM
IBMLink	IMS	MQSeries
MQSeries Three Tier	MVS/ESA	NetView
OS/2	OS/400	RACF
VSE/ESA		

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

C-bus is a trademark of Corollary, Inc.

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

Java and HotJava are trademarks of Sun Microsystems, Inc.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

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

adapter. An interface between MQSeries for MVS/ESA and TSO, IMS, CICS, or batch address spaces. An adapter is an attachment facility that enables applications to access MQSeries services.

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.

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.

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-defined format. In message queuing, application data in a message, which has a meaning defined by the user application. Contrast with *built-in format*.

application environment. The software facilities that are accessible by an application program. On the MVS platform, CICS and IMS are examples of application environments.

application queue. A queue used by an application.

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 open an MQSeries object.

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.

authorized program analysis report (APAR). A report of a problem caused by a suspected defect in a current, unaltered release of a program.

B

back out. An operation that reverses 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 *commit*.

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.

built-in format. In message queuing, application data in a message, which has a meaning defined by the queue manager. Synonymous with *in-built format*. Contrast with *application-defined format*.

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*.

© Copyright IBM Corp. 1994, 1998

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.

channel exit program. A user-written program that can be entered from one of a defined number of places during channel operation.

channel initiator. A component of MQSeries distributed queuing, which monitors the initiation queue to see when triggering criteria have been met and then starts the sender channel.

channel listener. A component of MQSeries distributed queuing, which monitors the network for a startup request and then starts the receiving channel.

checkpoint. A time when significant information is written on the log. Contrast with *syncpoint*.

CI. Control interval.

CICS transaction. In CICS, a unit of application processing, usually comprising one or more units of work.

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*.

coded character set identifier (CCSID). The name of a coded set of characters and their code point assignments.

command. In MQSeries, an instruction that can be carried out by the queue manager.

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.

connect. To provide a queue manager connection handle, which an application uses on subsequent MQI calls. The connection is made either by the MQCONN 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 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*.

D

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.

DCE principal. A user ID that uses the distributed computing environment.

DCI. Data conversion interface.

data-conversion service. A service that converts application data to the character set and encoding that are required by applications on other platforms.

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.

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.

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*.

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 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.

F

FAP. Formats and Protocols.

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)

format. In message queuing, a term used to identify the nature of application data in a message. See also *built-in format* and *application-defined format*.

Formats and Protocols (FAP). The MQSeries FAPs define how queue managers communicate with one another, and also how MQSeries clients communicate with server queue managers.

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)

G

get. In message queuing, to use the MQGET call to remove a message from a queue. See also *browse*.

H

handle. See *connection handle* and *object handle*.

heartbeat flow. A pulse that is passed from a sending MCA to a receiving MCA when there are no messages to send. The pulse unblocks the receiving MCA, which otherwise, would remain in a wait state until a message arrived or the disconnect interval expired.

heartbeat interval. The time, in seconds, that is to elapse between heartbeat flows.

I

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*.

in-built format. See *built-in format*.

initialization file. In MQSeries for AS/400, a file that contains two parameters; the TCP/IP listener port number and the maximum number of channels that can be current at a time. The file is called QMINI.

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.

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.

IPCS. Interactive Problem Control System.

L

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.

log. In MQSeries, a file recording the work done by queue managers while they receive, transmit, and deliver messages.

M

message. In message queuing applications, a communication sent between programs. See also *persistent message* and *nonpersistent message*. 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 and a receiver) 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.

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-retry. An option available to an MCA that is unable to deliver a message. The MCA can wait for a predefined amount of time and then try to send the message again.

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.

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 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. Contrast with *programmable command format (PCF)*.

N

namelist. An MQSeries for MVS/ESA object that contains a list of queue names.

nonpersistent message. A message that does not survive a restart of the queue manager. Contrast with *persistent message*.

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 (MVS/ESA only), or a storage class (MVS/ESA only).

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.

output parameter. A parameter of an MQI call in which the queue manager returns information when the call completes or fails.

P

PCF. Programmable command format.

PCF command. See *programmable command format*.

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.

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*.

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.

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.

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

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.

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.

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.

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.

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.

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.

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.

return-to-sender. An option available to an MCA that is unable to deliver a message. The MCA can send the message back to the originator.

rollback. Synonym for *back out*.

rules table. A control file containing one or more rules that the dead-letter queue handler applies to messages on the DLQ.

S

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.

shutdown. See *immediate shutdown*, *preemptive shutdown*, and *quiesced shutdown*.

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*.

SNA. Systems network architecture.

source queue manager. See *local queue manager*.

star-connected communications network. A network in which all nodes are connected to a central node.

store and forward. The temporary storing of packets, messages, or frames in a data network before they are retransmitted toward their destination.

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.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.

systems network architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of, networks.

T

target queue manager. See *remote queue manager*.

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*.

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). See *performance trace*.

transaction. See *unit of work* and *CICS transaction*.

transaction manager. A software unit that coordinates the activities of resource managers by managing global transactions and coordinating the decision to commit them or roll them back. V5.0 of MQSeries for AIX, HP-UX, OS/2 Warp, Sun Solaris, and Windows NT is a transaction manager.

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*.

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*.

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.

Index

A

AC* values 88, 323
alias queue 262
aliasing
  queue manager 260
  reply queue 260
AMQ3ECH4 sample program 364
AMQ3GBR4 sample program 359
AMQ3GET4 sample program 360
AMQ3INQ4 sample program 365
AMQ3PUT4 sample program 358
AMQ3REQ4 sample program 360
AMQ3SET4 sample program 367
AMQ3SRV4 sample program 368
AMQ3TRG4 sample program 368
ApplId
  attribute, process-definition attributes 262
ApplType
  attribute, process-definition attributes 263
AT* values 89
  ApplType field
    process-definition attributes 263
  TMAT field
    MQTM structure 153
  values of constants 323
attributes
  alias queue 262
  common to all queues 243
  local queue 246
  process definition 262
  queue manager 264
  remote queue, local definition of 260
AuthorityEvent attribute 266

B

BackoutRequeueQName attribute 247
BackoutThreshold attribute 248
BaseQName attribute 262
bibliography ix
BookManager xiii
BUFFER parameter
  MQGET call 184
  MQPUT call 218
  MQPUT1 call 228
BUFLEN parameter
  MQGET call 184
  MQPUT call 217
  MQPUT1 call 227
building your application 349
built-in formats 78

C

CA* values 195, 323
CALEN parameter
  MQINQ call 198
  MQSET call 238
calls
  conventions used 167
  detailed description
    MQCLOSE 169
    MQCONN 175
    MQDATACONVEXIT 398
    MQDISC 180
    MQGET 183
    MQINQ 194
    MQOPEN 204
    MQPUT 217
    MQPUT1 227
    MQSET 236
    MQXCNVC 403
CC* values 275, 324
ChannelAutoDef attribute 266
ChannelAutoDefEvent attribute 266
ChannelAutoDefExit attribute 266
CHRATR parameter
  MQINQ call 198
  MQSET call 238
CI* values 85, 324
CMLV* values 267, 325
CMPCOD parameter
  MQCLOSE call 171
  MQCONN call 176
  MQDISC call 180
  MQGET call 185
  MQINQ call 199
  MQOPEN call 209
  MQPUT call 218
  MQPUT1 call 228
  MQSET call 238
  MQXCNVC call 406
CO* values 169, 325
coded character set identifier 267
CodedCharSetId
  attribute, queue-manager attributes 267
CommandInputQName attribute 267
CommandLevel attribute 267
commitment control considerations 350
compiling 349
completion code 275

constants, values of 321, 345
accounting token (AC*) 323
application type (AT*) 323
backout hardening (QA*) 337
call identifier (MQ*) 324
character attribute selectors (CA*) 323
close options (CO*) 325
coded character set identifier (CS*) 324
command level (CMLV*) 325
completion codes (CC*) 324
connection handle (HC*) 330
convert-characters masks and factors (DCC*) 325
convert-characters options (DCC*) 325
correlation identifier (CI*) 324
data-conversion-exit parameter structure identifier (DX*) 326
data-conversion-exit parameter structure version (DX*) 326
data-conversion-exit response (XR*) 345
dead-letter header structure identifier (DL*) 326
dead-letter header version (DL*) 326
distribution header flags (DHF*) 326
distribution header structure identifier (DH*) 326
distribution header version (DH*) 326
distribution list support (DL*) 326
encoding (EN*) 327
encoding for binary integers (EN*) 327
encoding for floating-point numbers (EN*) 327
encoding for packed-decimal integers (EN*) 327
encoding masks (EN*) 327
event reporting (EV*) 327
event reporting (QSIE*) 337
expiry interval (EI*) 327
feedback (FB*) 327
format (FM*) 328
get message options (GM*) 329
get message options structure identifier (GM*) 329
get message options version (GM*) 329
group identifier (GI*) 329
group status (GS*) 330
IMS authenticator (IAU*) 331
IMS commit mode (ICM*) 331
IMS header flags (II*) 332
IMS header length (II*) 332
IMS header structure identifier (II*) 332
IMS header version (II*) 332
IMS security scope (ISS*) 332
IMS transaction instance identifier (ITI*) 332
IMS transaction state (ITS*) 332
Index type (IT*) 332
inhibit get (QA*) 337
inhibit put (QA*) 337
integer attribute selectors (IA*) 330
integer attribute value (IAV*) 331
lengths of character string and byte fields (LN*) 321
match options (MO*) 334

constants, values of *(continued)*
message delivery sequence (MS*) 333
message descriptor extension flags (MEF*) 333
message descriptor extension length (ME*) 333
message descriptor extension structure identifier (ME*) 333
message descriptor extension version (ME*) 333
message descriptor structure identifier (MD*) 333
message descriptor version (MD*) 333
message flags (MF*) 333
message identifier (MI*) 334
message type (MT*) 334
message-flags masks (MF*) 334
object descriptor length (OD*) 334
object descriptor structure identifier (OD*) 334
object descriptor version (OD*) 335
object handle (HO*) 330
object instance identifier (OII*) 335
object type (OT*) 335
open options (OO*) 335
original length (OL*) 335
persistence (PE*) 335
platform (PL*) 336
priority (PR*) 337
put message options (PM*) 336
put message options length (PM*) 336
put message options structure identifier (PM*) 336
put message options version (PM*) 336
put message record field flags (PF*) 336
queue definition type (QD*) 337
queue shareability (QA*) 337
queue type (QT*) 338
reason codes (RC*) 338
reference message header flags (RM*) 342
reference message header structure identifier (RM*) 342
reference message header version (RM*) 342
report options (RO*) 342
report-options masks (RO*) 343
segment status (SS*) 343
segmentation (SEG*) 343
syncpoint (SP*) 343
transmission queue header structure identifier 345
transmission queue header version (XQ*) 345
trigger controls (TC*) 343
trigger message (character format) structure identifier (TC*) 344
trigger message (character format) version (TC*) 344
trigger message structure identifier (TM*) 344
trigger message version (TM*) 344
trigger type (TT*) 344
undelivered-message header structure identifier (DL*) 326
undelivered-message header version (DL*) 326
usage (US*) 344

constants, values of *(continued)*
  wait interval (WI*) 344
conversion of report messages 390
conversion processing conventions 385
copy file – RPG programming language 8
copy files 349
CreationDate attribute 248
CreationTime attribute 248
CRTPGM 349
CRTRPGMOD 349
CRTRPGPGM 349
CS* values 77, 324
CurrentQDepth attribute 248

D

data conversion
  processing conventions 385
data conversion processing 385
data types, conventions used 3, 7
data types, detailed description
  elementary
    ILE 5
    MQBYTE 3
    MQBYTEn 4
    MQCHAR 4
    MQCHARn 4
    MQHCONN 5
    MQHOBJ 5
    MQLONG 5
    OPM 6
    overview 3
  structure
    MQDH structure 13
    MQDLH 19
    MQDXP 391
    MQGMO 27
    MQIIH 53
    MQMD 59
    MQMDE 104
    MQOD 110
    MQOR 118
    MQPMO 120
    MQPMR 137
    MQRMH 140
    MQRR 149
    MQTM 151
    MQTMC 156
    MQXQH 159
    overview of 7
DATLEN
  parameter, MQGET call 184
DCC* values 325
dead-letter header structure 19
DeadLetterQName attribute 268
DefinitionType attribute 249
DefInputOpenOption attribute 250
DefPersistence attribute 244
DefPriority attribute 244
DefXmitQName attribute 269
DH* values 14, 326
DHCNT field 16
DHCSI
  field
    MQDH 15
DHENC field 15
DHF* values 326
DHFLG field 15
DHFMT field 15
DHLEN field 14
DHORO field 16
DHPRF field 15
DHPRO field 16
DHSID field 14
DHVER field 14
DistLists attribute 250, 269
distribution header structure 13
distribution lists 250, 269
DL* values 21, 326
DLCSI 23
DLDM field 22
DLDQ field 22
DLENC field 22
DLFMT field 23
DLPAN field 24
DLPAT field 23
DLPD field 24
DLPT field 24
DLREA field 21
DLSID field 21
DLVER field 21
DX* values 326, 392
DXAOP field 393
DXCC field
  MQDXP structure 394
DXCSI
  field
    MQDXP structure 393
DXENC field
  MQDXP structure 393
DXHCN field 396
DXLEN
  field, MQDXP structure 393
DXREA field
  MQDXP structure 394
DXRES field
  MQDXP structure 395
DXSID field 392
DXVER field 392
DXXOP field 392

dynamic queue 204

E
EI* values 74, 327
EN* values 77, 327
Encoding field
  using 375
EnvData
  attribute process-definition attributes 263
EV* values 255, 256, 266, 269, 271, 272, 327

F
FB* values 21, 74, 327
FM* values 328
fonts in this book viii
formats built-in 78

G
get-message options structure 27
GI* values 93, 329
glossary 413
GM* values 27, 28, 329
GMGST field 49
GMMO field 47
GMO parameter 183
GMOPT field 28
GMRE1 field 50
GMRQN field 47
GMSEG field 50
GMSG1 field 47
GMSG2 field 47
GMSID field 27
GMSST field 50
GMVER field 27
GMWI field 46
GS* values 330

H
handle scope 176, 209
Handles 269
HardenGetBackout attribute 251
HC* values 330
HCONN parameter
  MQCLOSE call 169
  MQCONN call 176
  MQDISC call 180
  MQGET call 183
  MQINQ call 194
  MQOPEN call 204
  MQPUT call 217
  MQPUT1 call 227
  MQSET call 236
  MQXCNVC call 403
HCONN parameter *(continued)*
  scope 176
HO* values 330
HOBJ parameter
  MQCLOSE call 169
  MQGET call 183
  MQINQ call 194
  MQOPEN call 209
  MQPUT call 217
  MQSET call 236
  scope 209
HTML (Hypertext Markup Language) xiii
Hypertext Markup Language (HTML) xiii

I
IA* values 195, 237, 330
IACNT parameter
  MQINQ call 198
  MQSET call 237
IAU* values 331
IAV* values 198, 331
ICM* values 331
II* values 54, 332
IIAUT field 55
IICMT field 56
IICSI field 54
IIENC field 54
IIFLG field 54
IIFMT field
  MQIIH structure 54
IILEN field 54
IILTO field 55
IIMMN field 55
IIRFM field 55
IIRSV field 56
IISEC field 56
IISID field 54
IITID field 55
IITST field 56
INBUF parameter 399
Information Presentation Facility (IPF) xiii
InhibitEvent attribute 269
InhibitGet attribute 245
InhibitPut attribute 245
InitiationQName attribute 252
INLEN parameter 398
INTATR parameter
  MQINQ call 198
  MQSET call 237
IPF (Information Presentation Facility) xiii
ISS* values 332
IT* values 332
ITI* values 332
ITS* values 332

L

LN* values 321
LocalEvent attribute 269

M

MaxHandles attribute 269
MaxMsgLength attribute
  local-queue attributes 252
  queue-manager attributes 270
MaxPriority attribute 270
MaxQDepth attribute 252
MaxUncommittedMsgs attribute 270
MD* values 61, 333
MDACC field
  MQMD structure 87
MDAID field 88
MDAOD field 92
MDBOC field 85
MDCID field 85
MDCSI field 77
MDENC field 77
MDEXP field 72
MDFB field 74
MDFMT field 78
MDGID field 92
MDMFL field 95
MDMID field 83
MDMT field 71
MDOFF field 94
MDOLN field 99
MDPAN field 90
MDPAT field 89
MDPD field 90
MDPER field 82
MDPRI field 81
MDPT field 91
MDREP field 61
MDRM field 86
MDRQ field 85
MDSEQ field 94
MDSID field 61
MDUID field 87
MDVER field 61
ME* values 106, 333
MECSI field 107
MEENC field 107
MEF* values 333
MEFLG field 107
MEFMT field 107
MEGID field 107
MELEN field 107
MEMFL field 108
MEOFF field 108
MEOLN field 108
MESEQ field 108
MESID field 106
message descriptor extension structure 104
message descriptor structure 59
message order 188
MEVER field 106
MF* values 95, 333, 334
MI* values 84, 334
MO* values 334
MQ* values 324
MQBYTE 3
MQBYTEn 4
MQCHAR 4
MQCHARn 4
MQCLOSE 169
MQCONN 175
MQDATACONVEXIT 398
MQDH 13
MQDISC 180
MQDLH 19
MQDXP 391
MQDXP parameter 398
MQGET 183
MQGMO 27
MQHCONN 5
MQHOBJ 5
MQIIH 53
MQINQ 194
MQLONG 5
MQMD 59
MQMD parameter
  MQDATACONVEXIT call 398
MQMDE 104
MQOD 110
MQOPEN 204
MQOR 118
MQPMO 120
MQPMR 137
MQPUT 217
MQPUT1 227
MQRMH 140
MQRR 149
MQSeries for AS/400
  commitment control considerations 350
  syncpoint considerations with CICS for AS/400 351
  syncpoints 350
MQSeries publications ix
MQSET 236
MQTM 151
MQTMC 156
MQXCNVC 403
MQXQH 159
MS* values 253, 333
MsgDeliverySequence attribute 253

MSGDSC parameter
  MQGET call 183
  MQPUT call 217
  MQPUT1 call 227
MT* values 71, 334

N

notational conventions – RPG programming language 10

O

OBJDSC parameter
  MQOPEN call 204
  MQPUT1 call 227
object descriptor structure 110
object record structure 118
OD* values 111, 334, 335
ODAU field 113
ODDN field 112
ODIDC field 114
ODKDC field 113
ODMN field 112
ODON field 111
ODORO field 114
ODORP field 115
ODOT field 111
ODREC field 113
ODRRO field 114
ODRRP field 115
ODSID field 111
ODUDC field 113
ODVER field 111
OII* values 143, 335
OL* values 99, 335
OO* values 205, 250, 335
OpenInputCount attribute 254
OpenOutputCount attribute 254
OPTS parameter
  MQCLOSE call 169
  MQOPEN call 204
  MQXCNVC call 403
ordering of messages 188
ORMN field 118
ORON field 118
OT* values 111, 335
OUTBUF parameter 399
OUTLEN parameter 399

P

PE* values 82, 244, 335
PerformanceEvent attribute 271
persistence 244
PF* values 336
PL* values 271, 336
Platform attribute 271
PM* values 121, 336
PMCT field 130
PMIDC field 130
PMKDC field 130
PMO parameter
  MQPUT call 217
  MQPUT1 call 227
PMOPT field 121
PMPRF field 131
PMPRO field 132
PMPRP field 134
PMREC field 131
PMRMN field 131
PMRQN field 130
PMRRO field 133
PMRRP field 134
PMSID field 121
PMTO field 129
PMUDC field 130
PMVER field 121
PostScript format xiii
PR* values 81, 337
PRACC field 138
PRCID field 138
PRFB field 138
PRGID field 138
PRMID field 137
process definition attributes 262
ProcessDesc attribute 263
processing conventions 385
ProcessName
  attribute
    local-queue attributes 254
    process-definition attributes 264
publications
  MQSeries ix
  related xiv
put message record structure 137
put-message options structure 120

Q

QA* values 245, 257, 337
QD* values 249, 337
QDepthHighEvent attribute 254
QDepthHighLimit attribute 255
QDepthLowEvent attribute 255
QDepthLowLimit attribute 255
QDepthMaxEvent attribute 256
QDesc attribute 246
QMgrDesc attribute 271
QMgrName
  attribute, queue-manager attributes 271

QMNAME parameter
  MQCONN call 175
QName
  attribute, attributes common to all queues 246
QRPGLESRC 349
QServiceInterval attribute 256
QServiceIntervalEvent attribute 256
QSIE* values 256, 337
QT* values 246, 262, 338
QType attribute 246
queue attributes
  alias 262
  common to all queues 243
  local 246
  local definition of remote 260
  model 246
queue manager attributes 264
queue-manager aliasing 260
queue, dynamic 204

R

RC* values 76, 276, 338
reason codes
  alphabetic list 275
  numeric list 338
REASON parameter
  MQCLOSE call 171
  MQCONN call 177
  MQDISC call 180
  MQGET call 185
  MQINQ call 199
  MQOPEN call 210
  MQPUT call 218
  MQPUT1 call 228
  MQSET call 238
  MQXCNVC call 406
reference message header structure 140
RemoteEvent attribute 272
RemoteQMgrName
  attribute, remote-queue (local definition)
    attributes 260
RemoteQName
  attribute, remote-queue (local definition)
    attributes 261
reply queue aliasing 260
Report field
  using 379
report message conversion 390
response record structure 149
RetentionInterval attribute 257
return codes 275
RM* values 141, 142, 342
RMCSI field 142
RMDEL field 144
RMDEO field 144
RMDL field 145
RMDNL field 144
RMDNO field 144
RMDO field 145
RMDO2 field 146
RMENC field 142
RMFLG field 142
RMFMT field 142
RMLEN field 141
RMOII field 143
RMOT field 143
RMSEL field 143
RMSEO field 143
RMSID field 141
RMSNL field 143
RMSNO field 144
RMVER field 141
RO* values 62, 342, 343
RPG (ILE) sample programs 355
RPG programming language
  COPY file 8
  notational conventions 10
  structures 10, 349
RRCC field 149
RRREA field 149

S

sample programs 355
  browse 359
  echo 364
  get 360
  inquire 365
  preparing and running 357
  put 358
  request 360
  set 367
  trigger monitor 368
  trigger server 368
  using remote queues 369
  using triggering 361
scope, handles 176, 209
SEG* values 343
SELCNT parameter
  MQINQ call 194
  MQSET call 236
SELS parameter
  MQINQ call 194
  MQSET call 236
Shareability attribute 257
softcopy books xii
softcopy links viii
SP* values 272, 343
SRCBUF parameter 405

SRCCSI parameter 405
SRCLEN parameter 405
SS* values 343
StartStopEvent attribute 272
structures – RPG programming language 10, 349
syncpoint 272
  in CICS for AS/400 applications 351
  with MQSeries for AS/400 350
SyncPoint attribute 272

T

TC* values 258, 343, 344
TCAI field 157
TCAT field 157
TCED field 157
TCPN field 157
TCQN field 156
TCSID field 156
TCTD field 157
TCUD field 157
TCVER field 156
terminology viii
terminology used in this book 413
TGTBUF parameter 405
TGTCSI parameter 405
TGTLEN parameter 405
TM* values 152, 344
TMAI field 153
TMAT field 153
TMED field 154
TMPN field 153
TMQN field 152
TMSID field 152
TMTD field 153
TMUD field 154
TMVER field 152
transmission queue header structure 159
trigger message structure 151
TriggerControl attribute 257
TriggerData
  attribute, local-queue attributes 258
TriggerDepth attribute 258
triggering 257
TriggerInterval attribute 272
TriggerMsgPriority attribute 258
TriggerType attribute 259
TT* values 259, 344
type styles in this book viii

U

Uncommitted messages 270
US* values 259, 344
Usage attribute 259
UserData
  attribute process-definition attributes 264

W

WI* values 46, 344
Windows Help xiii

X

XmitQName attribute, remote-queue (local definition) attributes 261
XQ* values 162, 345
XQMD field 162
XQRQ field 162
XQRQM field 162
XQSID field 162
XQVER field 162
XR* values 345

Sending your comments to IBM

MQSeries for AS/400

Application Programming Reference (RPG)

SC33-1957-00

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 request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should 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, use the Readers' Comment Form
- 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: WINVMD(IDRCF)
  - Internet: idrcf@winvmd.vnet.ibm.com

Whichever you use, ensure that you include:

- The publication number and title
- The page number or topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Readers' Comments

MQSeries for AS/400

Application Programming Reference (RPG)

SC33-1957-00

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

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.

Be sure to print your name and address below if you would like a reply.

Name

Address

Company or Organization

Telephone

Email

MQSeries for AS/400

RPG Reference SC33-1957-00

You can send your comments POST FREE on this form from any one of these countries:

Australia, Belgium, Bermuda, Cyprus, Denmark, Finland, France, Germany, Greece, Hong Kong, Iceland, Israel, Italy, Luxembourg, Monaco, Netherlands, New Zealand, Norway, Portugal, Republic of Ireland, Singapore, Spain, Sweden, Switzerland, United Arab Emirates, United States of America

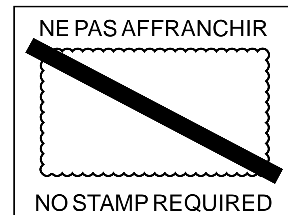
If your country is not listed here, your local IBM representative will be pleased to forward your comments to us. Or you can pay the postage and send the form direct to IBM (this includes mailing in the U.K.).

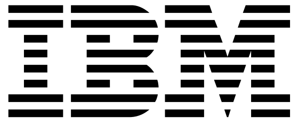
1 Cut along this line

2 Fold along this line

By air mail
Par avion

IBRS/CCRI NUMBER: PHQ - D/1348/SO





REPONSE PAYEE
GRANDE-BRETAGNE

IBM United Kingdom Laboratories
Information Development Department (MP095)
Hursley Park,
WINCHESTER, Hants
SO21 2ZZ United Kingdom

3 Fold along this line

From: Name ____________________

Company or Organization ____________________

Address ____________________

____

EMAIL ____________________

Telephone ____________________

1 Cut along this line

4 Fasten here with adhesive tape

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.