

WebSphere MQ



Application Programming Reference

Version 7.0

WebSphere MQ



Application Programming Reference

Version 7.0

Note

Before using this information and the product it supports, be sure to read the general information under notices at the back of this book.

Second edition (January 2009)

This edition of the book applies to the following products:

- IBM WebSphere MQ, Version 7.0
- IBM WebSphere MQ for z/OS, Version 7.0

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

© **Copyright International Business Machines Corporation 1994, 2009.**

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

Contents

Tables	ix
-------------------------	-----------

Chapter 1. Data type descriptions. **1**

Introduction	1
Elementary data types	1
Structure data types – introduction	17
C programming	20
COBOL programming	23
System/390 assembler programming	26
MQAIR – Authentication information record	29
Overview for MQAIR	29
Fields for MQAIR	29
Initial values and language declarations for MQAIR.	32
MQBMHO – Buffer to message handle options	33
Overview for MQBMHO	33
Fields for MQBMHO	34
Initial values and language declarations for MQBMHO.	34
MQBO – Begin options	35
Overview for MQBO	36
Fields for MQBO	36
Initial values and language declarations for MQBO	37
MQCBC – Callback context	38
Overview for MQCBC.	38
Fields for MQCBC	38
Initial values and language declarations for MQCBC	44
MQCBD – Callback descriptor	46
Overview for MQCBD.	46
Fields for MQCBD	47
Initial values and language declarations for MQCBD	51
MQCHARV - Variable Length String	53
Overview for MQCHARV	53
Fields for MQCHARV	54
Initial values and language declarations for MQCHARV	55
Redefinition of MQCCSI_APPL.	56
MQCIH – CICS bridge header	57
Overview for MQCIH	58
Fields for MQCIH	59
Initial values and language declarations for MQCIH	69
MQCMHO – Create-message options.	74
Overview for MQCMHO.	75
Fields for MQCMHO	75
Initial values and language declarations for MQCMHO	77
MQCNO – Connect options	77
Overview for MQCNO	78
Fields for MQCNO.	78
Initial values and language declarations for MQCNO	90

MQCSP – Security parameters	92
Overview for MQCSP	92
Fields for MQCSP	93
Initial values and language declarations for MQCSP.	94
MQCTLO – Control callback options structure.	96
Overview for MQCTLO	96
Fields for MQCTLO	97
Initial values and language declarations for MQCTLO	98
MQDH – Distribution header	99
Overview for MQDH.	100
Fields for MQDH	101
Initial values and language declarations for MQDH	104
MQDLH – Dead-letter header	106
Overview for MQDLH	107
Fields for MQDLH	109
Initial values and language declarations for MQDLH	113
MQDMHO – Delete message handle options	116
Overview for MQDMHO	116
Fields for MQDMHO.	116
Initial values and language declarations for MQDMHO	117
MQDMPO – Delete message property options	118
Overview for MQDMPO	118
Fields for MQDMPO	118
Initial values and language declarations for MQDMPO	119
MQEPH – Embedded PCF header	120
Overview for MQEPH	121
Fields for MQEPH.	121
Initial values and language declarations for MQEPH	124
MQGMO – Get-message options	126
Overview for MQGMO	127
Fields for MQGMO	127
Initial values and language declarations for MQGMO.	160
MQIIH – IMS information header	163
Overview for MQIIH.	164
Fields for MQIIH	164
Initial values and language declarations for MQIIH	168
MQIMPO – Inquire message property options	171
Overview for MQIMPO	171
Fields for MQIMPO	171
Initial values and language declarations for MQIMPO.	178
MQMD – Message descriptor	181
Overview for MQMD	182
Fields for MQMD	183
Initial values and language declarations for MQMD	235
MQMDE – Message descriptor extension	239

Overview for MQMDE	240	Fields for MQSD	334
Fields for MQMDE	242	Using topic strings	351
Initial values and language declarations for MQMDE	244	Initial values and language declarations for MQSD.	352
MQMHBO – Message handle to buffer options	247	MQSMPO – Set message property options.	355
Overview for MQMHBO	247	Overview for MQSMPO.	355
Fields for MQMHBO	247	Fields for MQSMPO	356
Initial values and language declarations for MQMHBO	248	Initial values and language declarations for MQSMPO	357
MQOD – Object descriptor	249	MQSRO - Subscription request options	359
Overview for MQOD.	250	Overview for MQSRO	359
Fields for MQOD	251	Fields for MQSRO.	359
Initial values and language declarations for MQOD	261	Initial values and language declarations for MQSRO	360
MQOR – Object record	266	MQSTS – Status reporting structure	361
Overview for MQOR.	266	Overview for MQSTS.	362
Fields for MQOR	266	Fields for MQSTS	362
Initial values and language declarations for MQOR	267	Initial values and language declarations for MQSTS	364
MQPD – Property descriptor	267	MQTM – Trigger message	366
Overview for MQPD	268	Overview for MQTM.	366
Fields for MQPD	268	Fields for MQTM	368
Initial values and language declarations for MQPD	271	Initial values and language declarations for MQTM	372
MQPMO – Put-message options	272	MQTMC2 – Trigger message 2 (character format)	373
Overview for MQPMO	273	Overview for MQTMC2.	374
Fields for MQPMO	274	Fields for MQTMC2	374
Initial values and language declarations for MQPMO	296	Initial values and language declarations for MQTMC2	376
MQPMR – Put-message record	300	MQWIH – Work information header	378
Overview for MQPMR	300	Overview for MQWIH	378
Fields for MQPMR	301	Fields for MQWIH	378
Initial values and language declarations for MQPMR	302	Initial values and language declarations for MQWIH	381
MQRFH – Rules and formatting header	303	MQXP – Exit parameter block	383
Overview for MQRFH	303	Overview for MQXP	383
Fields for MQRFH.	303	Fields for MQXP	383
Initial values and language declarations for MQRFH	306	Language declarations	386
MQRFH2 – Rules and formatting header 2	308	MQXQH – Transmission-queue header	387
Overview for MQRFH2	308	Overview for MQXQH	387
Fields for MQRFH2	308	Fields for MQXQH	390
Initial values and language declarations for MQRFH2.	313	Initial values and language declarations for MQXQH	392
MQRMH – Reference message header	315		
Overview for MQRMH	316	Chapter 2. Function calls 395	
Fields for MQRMH	317	Call descriptions	395
Initial values and language declarations for MQRMH	322	Conventions used in the call descriptions	395
MQRR – Response record	325	Using the calls in the C language.	397
Overview for MQRR	326	MQBACK – Back out changes	397
Fields for MQRR	326	Syntax for MQBACK	397
Initial values and language declarations for MQRR.	326	Parameters for MQBACK	398
MQSCO – SSL configuration options	327	Usage notes for MQBACK	399
Overview for MQSCO	328	Language invocations for MQBACK.	400
Fields for MQSCO.	328	MQBEGIN – Begin unit of work	401
Initial values and language declarations for MQSCO	332	Syntax for MQBEGIN	401
MQSD - Subscription descriptor	333	Parameters for MQBEGIN	402
Overview for MQSD	334	Usage notes for MQBEGIN.	403
		Language invocations for MQBEGIN	404
		MQBUFMH - Convert buffer into message handle	405
		Syntax for MQBUFMH	405
		Parameters for MQBUFMH.	405

Usage notes for MQBUFMH	408	Parameters for MQINQ	481
Language invocations for MQBUFMH	408	Usage notes for MQINQ.	492
MQCB – Manage callback	409	Language invocations for MQINQ	493
Syntax for MQCB	410	MQINQMP - Inquire message property.	495
Parameters for MQCB	410	Syntax for MQINQMP	495
Usage notes for MQCB	417	Parameters for MQINQMP	495
Language invocations for MQCB	418	Language invocations for MQINQMP	500
MQCB_FUNCTION – Callback function	419	MQMHBUF - Convert message handle into buffer	501
Syntax for MQCB_FUNCTION	420	Syntax for MQMHBUF	501
Parameters for MQCB_FUNCTION	420	Parameters for MQMHBUF.	501
Usage notes for MQCB_FUNCTION.	421	Usage notes for MQMHBUF	504
Language invocations for MQCB_FUNCTION	422	Language invocations for MQMHBUF	504
MQCLOSE – Close object	422	MQOPEN – Open object	505
Syntax for MQCLOSE	423	Syntax for MQOPEN	506
Parameters for MQCLOSE	423	Parameters for MQOPEN	506
Usage notes for MQCLOSE.	429	Usage notes for MQOPEN	517
Language invocations for MQCLOSE	430	Language invocations for MQOPEN.	523
MQCMIT – Commit changes	431	MQPUT – Put message	524
Syntax for MQCMIT	431	Syntax for MQPUT	524
Parameters for MQCMIT	432	Parameters for MQPUT	524
Usage notes for MQCMIT	433	Usage notes for MQPUT	533
Language invocations for MQCMIT	434	Language invocations for MQPUT	538
MQCONN – Connect queue manager	435	MQPUT1 – Put one message	539
Syntax for MQCONN	436	Syntax for MQPUT1	539
Parameters for MQCONN	436	Parameters for MQPUT1	539
Usage notes for MQCONN.	441	Usage notes for MQPUT1	547
Language invocations for MQCONN	443	Language invocations for MQPUT1	548
MQCONNX – Connect queue manager (extended)	444	MQSET – Set object attributes	550
Syntax for MQCONNX	444	Syntax for MQSET	550
Parameters for MQCONNX	444	Parameters for MQSET	550
Usage notes for MQCONNX	446	Usage notes for MQSET	554
Language invocations for MQCONNX	446	Language invocations for MQSET	555
MQCRTMH – Create message handle	447	MQSETMP – Set message property	556
Syntax for MQCRTMH	447	Syntax for MQSETMP	556
Parameters for MQCRTMH.	447	Parameters for MQSETMP	557
Language invocations for MQCRTMH	450	Language invocations for MQSETMP	560
MQCTL – Control callback	450	MQSTAT – Retrieve status information.	561
Syntax for MQCTL	450	Syntax for MQSTAT	561
Parameters for MQCTL	451	Parameters for MQSTAT.	561
Usage notes for MQCTL.	456	Usage notes for MQSTAT	563
Language invocations for MQCTL	457	Language invocations for MQSTAT	563
MQDISC – Disconnect queue manager	457	MQSUB - Register subscription	564
Syntax for MQDISC	458	Syntax for MQSUB	564
Parameters for MQDISC.	458	Parameters for MQSUB	564
Usage notes for MQDISC	459	Usage notes for MQSUB.	569
Language invocations for MQDISC	460	Language invocations for MQSUB	570
MQDLTMH – Delete message handle	461	MQSUBRQ - Subscription request	571
Syntax for MQDLTMH	461	Syntax for MQSUBRQ	571
Parameters for MQDLTMH.	461	Parameters for MQSUBRQ	571
Language invocations for MQDLTMH	463	Usage notes for MQSUBRQ	572
MQDLTMP - Delete message property	464	Language invocations for MQSUBRQ	573
Syntax for MQDLTMP	464		
Parameters for MQDLTMP	464		
Language invocations for MQDLTMP	466		
MQGET – Get message	466		
Syntax for MQGET	467		
Parameters for MQGET	467		
Usage notes for MQGET.	474		
Language invocations for MQGET	479		
MQINQ – Inquire object attributes	480		
Syntax for MQINQ	481		
		Chapter 3. Attributes of objects	575
		Attributes for the queue manager	575
		AccountingConnOverride (MQLONG)	578
		AccountingInterval (MQLONG)	578
		AdoptNewMCACheck (MQLONG)	579
		AdoptNewMCAType (MQLONG)	579
		AlterationDate (MQCHAR12)	579
		AlterationTime (MQCHAR8)	580
		AuthorityEvent (MQLONG)	580

BridgeEvent (MQLONG)	580		PubSubMode (MQLONG)	605
ChannelAutoDef (MQLONG)	580		QMgrDesc (MQCHAR64)	606
ChannelAutoDefEvent (MQLONG)	581		QMgrIdentifier (MQCHAR48)	606
ChannelAutoDefExit (MQCHARn)	581		QMgrName (MQCHAR48)	606
ChannelEvent (MQLONG)	581		QSGName (MQCHAR4)	607
ChannelInitiatorControl (MQLONG)	582		QueueAccounting (MQLONG)	607
ChannelMonitoring (MQLONG)	582		QueueMonitoring (MQLONG)	607
ChannelStatistics (MQLONG)	583		QueueStatistics (MQLONG)	608
ChinitAdapters (MQLONG)	583		ReceiveTimeout (MQLONG)	608
ChinitDispatchers (MQLONG)	584		ReceiveTimeoutMin (MQLONG)	609
ChinitTraceAutoStart (MQLONG)	584		ReceiveTimeoutType (MQLONG)	609
ChinitTraceTableSize (MQLONG)	584		RemoteEvent (MQLONG)	610
ClusterSenderMonitoringDefault (MQLONG)	585		RepositoryName (MQCHAR48)	610
ClusterSenderStatistics (MQLONG)	585		RepositoryNameList (MQCHAR48)	610
ClusterWorkloadData (MQCHAR32)	586		ScyCase(MQCHAR8)	610
ClusterWorkloadExit (MQCHARn)	586		SharedQMgrName (MQLONG)	611
ClusterWorkloadLength (MQLONG)	586		SSLEvent (MQLONG)	611
CLWLMRUChannels (MQLONG)	587		SSLFIPSRequired (MQLONG)	611
CLWLUseQ (MQLONG)	587		SSLKeyResetCount (MQLONG)	612
CodedCharSetId (MQLONG)	587		StartStopEvent (MQLONG)	612
CommandEvent (MQLONG)	587		StatisticsInterval (MQLONG)	613
CommandInputQName (MQCHAR48)	588		SyncPoint (MQLONG)	613
CommandLevel (MQLONG)	588		TCPChannels (MQLONG)	613
CommandServerControl (MQLONG)	591		TCPKeepAlive (MQLONG)	613
ConfigurationEvent (MQLONG)	591		TCPName (MQCHAR8)	614
DeadLetterQName (MQCHAR48)	592		TCPStackType (MQLONG)	614
DefXmitQName (MQCHAR48)	593		TraceRouteRecording (MQLONG)	614
DistLists (MQLONG)	593		TriggerInterval (MQLONG)	614
DNSGroup (MQCHAR18)	593		TriggerInterval (MQLONG)	615
DNSWLM (MQLONG)	593		Attributes for queues	615
ExpiryInterval (MQLONG)	594		AlterationDate (MQCHAR12)	619
IGQPputAuthority (MQLONG)	594		AlterationTime (MQCHAR8)	620
IGQUserId (MQLONG)	595		BackoutRequeueQName (MQCHAR48)	620
InhibitEvent (MQLONG)	595		BackoutThreshold (MQLONG)	620
IntraGroupQueueing (MQLONG)	595		BaseQName (MQCHAR48)	621
IPAddressVersion (MQLONG)	596		BaseType (MQCFIN)	621
ListenerTimer (MQLONG)	596		CFStrucName (MQCHAR12)	621
LocalEvent (MQLONG)	597		ClusterName (MQCHAR48)	622
LoggerEvent (MQLONG)	597		ClusterNameList (MQCHAR48)	622
LUGroupName (MQCHAR8)	597		CLWLQueuePriority (MQLONG)	622
LUName (MQCHAR8)	598		CLWLQueueRank (MQLONG)	622
LU62ARMSuffix (MQCHAR2)	598		CLWLUseQ (MQLONG)	623
LU62Channels (MQLONG)	598		CreationDate (MQCHAR12)	623
MaxActiveChannels (MQLONG)	598		CreationTime (MQCHAR8)	623
MaxChannels (MQLONG)	598		CurrentQDepth (MQLONG)	624
MaxHandles (MQLONG)	599		DefaultPutResponse (MQLONG)	624
MaxMsgLength (MQLONG)	599		DefBind (MQLONG)	625
MaxPriority (MQLONG)	599		DefinitionType (MQLONG)	625
MaxUncommittedMsgs (MQLONG)	600		DefInputOpenOption (MQLONG)	626
MQIAccounting (MQLONG)	600		DefPersistence (MQLONG)	627
MQIStatistics (MQLONG)	601		DefPriority (MQLONG)	627
MsgMarkBrowseInterval (MQLONG)	601		DefReadAhead (MQLONG)	628
OutboundPortMax (MQLONG)	602		DefPResp (MQLONG)	629
OutboundPortMin (MQLONG)	602		DistLists (MQLONG)	629
PerformanceEvent (MQLONG)	602		HardenGetBackout (MQLONG)	630
Platform (MQLONG)	602		IndexType (MQLONG)	630
PropertyControl (MQLONG)	603		InhibitGet (MQLONG)	633
PubSubNPInputMsg (MQLONG)	604		InhibitPut (MQLONG)	634
PubSubNPResponse (MQLONG)	604		InitiationQName (MQCHAR48)	634
PubSubMaxMsgRetryCount (MQLONG)	605		MaxMsgLength (MQLONG)	634
PubSubSyncPoint (MQLONG)	605		MaxQDepth (MQLONG)	635

MsgDeliverySequence (MQLONG)	636
NonPersistentMessageClass (MQLONG)	637
OpenInputCount (MQLONG)	637
OpenOutputCount (MQLONG)	637
ProcessName (MQCHAR48)	638
PropertyControl (MQLONG)	638
QDepthHighEvent (MQLONG)	639
QDepthHighLimit (MQLONG)	639
QDepthLowEvent (MQLONG)	640
QDepthLowLimit (MQLONG)	640
QDepthMaxEvent (MQLONG)	641
QDesc (MQCHAR64)	641
QName (MQCHAR48)	641
QServiceInterval (MQLONG)	642
QServiceIntervalEvent (MQLONG)	642
QSGDisp (MQLONG)	643
QueueAccounting (MQCHAR12)	644
QueueMonitoring (MQLONG)	644
QueueStatistics (MQCHAR12)	645
QType (MQLONG)	645
RemoteQMgrName (MQCHAR48)	645
RemoteQName (MQCHAR48)	646
RetentionInterval (MQLONG)	646
Scope (MQLONG)	647
Shareability (MQLONG)	648
StorageClass (MQCHAR8)	648
TriggerControl (MQLONG)	648
TriggerData (MQCHAR64)	649
TriggerDepth (MQLONG)	649
TriggerMsgPriority (MQLONG)	649
TriggerType (MQLONG)	650
Usage (MQLONG)	650
XmitQName (MQCHAR48)	651
Attributes for namelists	651
AlterationDate (MQCHAR12)	652
AlterationTime (MQCHAR8)	652
NameCount (MQLONG)	652
NamelistDesc (MQCHAR64)	652
NamelistName (MQCHAR48)	653
NamelistType (MQLONG)	653
Names (MQCHAR48xNameCount)	653
QSGDisp (MQLONG)	653
Attributes for process definitions	654
AlterationDate (MQCHAR12)	654
AlterationTime (MQCHAR8)	654
ApplId (MQCHAR256)	655
ApplType (MQLONG)	655
EnvData (MQCHAR128)	657
ProcessDesc (MQCHAR64)	657
ProcessName (MQCHAR48)	658
QSGDisp (MQLONG)	658
UserData (MQCHAR128)	658
Chapter 4. Return codes	661
Completion codes	661
Reason codes	661
Chapter 5. Rules for validating MQI options	663
MQOPEN call	663

MQPUT call	663
MQPUT1 call	664
MQGET call	664
MQCLOSE call	665
MQSUB call	665

Chapter 6. Machine encodings 667

Binary-integer encoding	667
Packed-decimal-integer encoding	668
Floating-point encoding	668
Constructing encodings	669
Analyzing encodings	669
Using bit operations	669
Using arithmetic	669
Summary of machine architecture encodings	670

Chapter 7. Report options and message flags 671

Structure of the report field	671
Analyzing the report field	673
Using bit operations	673
Using arithmetic	673
Structure of the message-flags field	674

Chapter 8. Data conversion 677

Conversion processing	677
Processing conventions	678
Conversion of report messages	683
MQDXP – Data-conversion exit parameter	684
Overview	684
Fields	685
C declaration	690
COBOL declaration (i5/OS only)	690
System/390 assembler declaration	690
MQXCNVC – Convert characters	691
Syntax	691
Parameters	691
C invocation	696
COBOL invocation (i5/OS only)	697
System/390 assembler invocation	697
MQ_DATA_CONV_EXIT – Data conversion exit	697
Syntax	698
Parameters	698
Usage notes	699
C invocation	702
COBOL invocation (i5/OS only)	702
System/390 assembler invocation	702

Chapter 9. Properties specified as MQRFH2 elements 703

Mapping property data types to MQRFH2 data types	703
Supported MQRFH2 folders	703
Generation of MQRFH2 headers	706
MQRFH2 folder restrictions	706
MQRFH2 element name conflicts	707
Mapping from property names to MQRFH2 folder and element names	707
Mapping property descriptor fields into MQRFH2 headers	708

I	MQRFH2 headers that are not valid	710
---	---------------------------------------------	-----

Chapter 10. Code page conversion 713

	Codeset names and CCSIDs	713
	National languages	714
	US English	715
	German	716
	Danish and Norwegian	716
	Finnish and Swedish	717
	Italian	718
	Spanish	719
	UK English /Gaelic	719
	French.	720
	Multilingual.	720
	Portuguese	721
	Icelandic	721
	Eastern European languages	722
	Cyrillic	723
	Estonian	724
	Latvian and Lithuanian	725
	Ukrainian	726
	Greek	726
	Turkish	727
	Hebrew	727
	Arabic.	728
	Farsi	729
	Urdu	729

	Thai	730
	Lao.	730
	Vietnamese	730
	Japanese Latin SBCS	731
	Japanese Katakana SBCS	732
	Japanese Kanji/ Latin Mixed	734
	Japanese Kanji/ Katakana Mixed.	735
	Korean	737
	Simplified Chinese	737
	Traditional Chinese	738
	z/OS conversion support	739
	i5/OS conversion support	757
	Unicode conversion support	757
	WebSphere MQ AIX support for Unicode	758
	WebSphere MQ HP-UX support for Unicode	758
	WebSphere MQ for Windows, Solaris, and Linux support for Unicode	758
	i5/OS support for Unicode.	759
	WebSphere MQ for z/OS support for Unicode	759

Notices 761

Index 765

Sending your comments to IBM . . . 775

Tables

1. Structure data types used on MQI calls (or exit functions):	17	49. Queue-manager action when MQMDE specified on MQPUT or MQPUT1 for MQMDE	241
2. Structure data types used in message data:	18	50. Initial values of fields in MQMDE for MQMDE	244
3. C header files	20	51. Fields in MQMHBO	247
4. COBOL COPY files	23	52. Initial values of fields in MQMHBO	248
5. Assembler macros	26	53. Fields in MQOR	266
6. Fields in MQAIR.	29	54. Initial values of fields in MQOR for MQOR	267
7. Initial values of fields in MQAIR for MQAIR	32	55. Fields in MQPD.	267
8. Fields in MQBMHO.	33	56. Initial values of fields in MQPD	271
9. Initial values of fields in MQBMHO	35	57. MQPMO structure	272
10. Fields in MQBO	36	58. Reply message handle transformation	276
11. Initial values of fields in MQBO for MQBO	37	59. Report message handle transformation	277
12. Fields in MQCBC	38	60. Source of user data	278
13. Initial values of fields in MQCBC	44	61. MQPUT options relating to messages in groups and segments of logical messages	284
14. Fields in MQCBD	46	62. Outcome when MQPUT or MQCLOSE call is not consistent with group and segment information	286
15. Initial values of fields in MQCBD	51	63. Initial values of fields in MQPMO	296
16. Fields in MQCIH.	57	64. Fields in MQPMR	300
17. Contents of error information fields in MQCIH structure for MQCIH	59	65. Initial values of fields in MQRFH for MQRFH	306
18. Initial values of fields in MQCIH for MQCIH	69	66. Initial values of fields in MQRFH2 for MQRFH2	313
19. Fields in MQCMHO	74	67. Fields in MQRMH	315
20. Initial values of fields in MQCMHO	77	68. Initial values of fields in MQRMH for MQRMH	322
21. Fields in MQCNO	77	69. Fields in MQRR.	325
22. Initial values of fields in MQCNO for MQCNO	90	70. Initial values of fields in MQRR for MQRR	326
23. Fields in MQCSP.	92	71. Fields in MQSCO	327
24. Initial values of fields in MQCSP for MQCSP	94	72. Initial values of fields in MQSCO	332
25. Fields in MQCTLO	96	73. Fields in QSMPO	355
26. Initial values of fields in MQCTLO.	98	74. Initial values of fields in QSMPO	357
27. Fields in MQDH	99	75. Fields in MQSTS	361
28. Initial values of fields in MQDH for MQDH	104	76. Initial values of fields in MQSTS	364
29. Fields in MQDLH	106	77. Fields in MQTM	366
30. Initial values of fields in MQDLH for MQDLH	113	78. Initial values of fields in MQTM for MQTM	372
31. Fields in MQDMHO	116	79. Fields in MQTMC2	374
32. Initial values of fields in MQDMHO	117	80. Initial values of fields in MQTMC2 for MQTMC2.	376
33. Fields in MQDMPO	118	81. Fields in MQWIH	378
34. Initial values of fields in MQDPMO	119	82. Initial values of fields in MQWIH for MQWIH	381
35. Fields in MQEPH	121	83. Fields in MQXP.	383
36. Initial values of fields in MQDH	122	84. Fields in MQXQH	387
37. Initial values of fields in MQEPH for MQEPH	124	85. Initial values of fields in MQXQH for MQXQH	392
38. Fields in MQGMO	126	86. MQCTL verb definitions	417
39. MQGET options relating to messages in groups and segments of logical messages	149	87. Parameter names used by MQCB_FUNCTION call	420
40. Outcome when MQGET or MQCLOSE call is not consistent with group and segment information	151	88. Scope of nonshared handles on various platforms	439
41. Initial values of fields in MQGMO for MQGMO	160	89. MQGET options permitted when read ahead is enabled.	478
42. Fields in MQIIH	163	90. MQINQ attribute selectors for queues	482
43. Initial values of fields in MQIIH for MQIIH	168		
44. Fields in MQIMPO.	171		
45. Initial values of fields in MQIPMO	179		
46. Fields in MQMD	181		
47. Initial values of fields in MQMD for MQMD	235		
48. Fields in MQMDE	239		

91. MQINQ attribute selectors for namelists	484	99. Attributes for namelists	651
92. MQINQ attribute selectors for process definitions	484	100. Attributes for process definitions	654
93. MQINQ attribute selectors for the queue manager	485	101. Summary of encodings for machine architectures	670
94. MQSET attribute selectors for queues	551	102. Fields in MQDXP	684
95. Attributes for the queue manager	575	103.	709
96. Attributes for queues	617	104.	709
97. Recommended or required values of queue index type when MQGMO_LOGICAL_ORDER not specified	632	105.	709
98. Recommended or required values of queue index type when MQGMO_LOGICAL_ORDER specified	632	106. Codeset names and CCSIDs.	714
		107. WebSphere MQ for z/OS CCSID conversion support	739

Chapter 1. Data type descriptions

Introduction

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

Elementary data types

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

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

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

- MQBOOL - Boolean
- MQBYTE - Byte
- MQBYTEn - String of n bytes
- MQCHAR - Single-byte character
- MQCHARn - String of n single-byte characters
- MQFLOAT32 - 32-bit floating-point number
- MQFLOAT64 - 64-bit floating-point number
- MQHCONFIG - Configuration handle
- MQHCONN - Connection handle
- MQHMSG - A message handle that gives access to a message
- MQHOBJ - Object handle
- MQINT8 - 8-bit integer
- MQINT16 - 16-bit integer
- MQINT32 - 32-bit integer
- MQINT64 - 64-bit integer
- MQLONG - Long integer
- MQPID - Process Id
- MQPTR - Pointer
- MQTID - Thread Id
- MQUINT8 - 8-bit unsigned integer
- MQUINT16 - 16-bit unsigned integer
- MQUINT32 - 32-bit unsigned integer
- MQUINT64 - 64-bit unsigned integer
- MQULONG - unsigned long integer
- PMQACH - Pointer to a data structure of type MQACH
- PMQAIR - Pointer to a data structure of type MQAIR
- PMQAXC - Pointer to a data structure of type MQAXC
- PMQAXP - Pointer to a data structure of type MQAXP
- PMQBMHO - Pointer to a data structure of type MQBMHO
- PMQBO - Pointer to a data structure of type MQBO

- PMQBOOL – Pointer to a data type of MQBOOL
- PMQBYTE – Pointer to data of type MQBYTE
- PMQBYTEn – Pointer to a data type of MQBYTEn, where n can be 8, 16, 24, 32, 40, 128
- PMQCBC – Pointer to a data structure of type MQCBC
- PMQCBD – Pointer to a data structure of type MQCBD
- PMQCHAR – Pointer to a data structure of type MQCHAR
- PMQCHARV – Pointer to a data structure of type MQCHARV
- PMQCHARn – Pointer to a data type of MQCHARn, where n can be 4, 8, 12, 20, 28, 32, 48, 64, 128, 256, 264
- PMQCIH – Pointer to a data structure of type MQCIH
- PMQCMHO – Pointer to a data structure of type MQCMHO
- PMQCNO – Pointer to a data structure of type MQCNO
- PMQCSP – Pointer to a data structure of type MQCSP
- PMQCTLO – Pointer to a data structure of type MQCTLO
- PMQDH – Pointer to a data structure of type MQDH
- PMQDHO – Pointer to a data structure of type MQDHO
- PMQDLH – Pointer to a data structure of type MQDLH
- PMQDMHO – Pointer to a data structure of type MQDMHO
- PMQDMPO – Pointer to a data structure of type MQDMPO
- PMQEPPH – Pointer to a data structure of type MQEPPH
- PMQFLOAT32 – Pointer to a data type of MQFLOAT32
- PMQFLOAT64 – Pointer to a data type of MQFLOAT64
- PMQFUNC – Pointer to a function
- PMQGMO – Pointer to a data structure of type MQGMO
- PMQHCONFIG – Pointer to a data type of MQHCONFIG
- PMQHCONN – Pointer to a data type of MQHCONN
- PMQHMSG – Pointer to a data type of MQHMSG
- PMQHOBJ – Pointer to a data type of MQHOBJ
- PMQIIH – Pointer to a data structure of type MQIIH
- PMQIMPO – Pointer to a data structure of type MQIMPO
- PMQINT8 – Pointer to a data type of MQINT8
- PMQINT16 – Pointer to a data type of MQINT16
- PMQINT32 – Pointer to a data type of MQINT32
- PMQINT64 – Pointer to a data type of MQINT64
- PMQLONG – Pointer to a data type of MQLONG
- PMQMD – Pointer to a data structure of type MQMD
- PMQMDE – Pointer to a data structure of type MQMDE
- PMQMD1 – Pointer to a data structure of type MQMD1
- PMQMHBO – Pointer to a data structure of type MQMHBO
- PMQOD – Pointer to a data structure of type MQOD
- PMQOR – Pointer to a data structure of type MQOR
- PMQPD – Pointer to a data structure of type MQPD
- PMQPID – Pointer to a process ID
- PMQPMO – Pointer to a data structure of type MQPMO

- PMQPTR – Pointer to a data type of MQPTR
- PMQRFH – Pointer to a data structure of type MQRFH
- PMQRFH2 – Pointer to a data structure of type MQRFH2
- PMQRMH – Pointer to a data structure of type MQRMH
- PMQRR – Pointer to a data structure of type MQRR
- PMQSCO – Pointer to a data structure of type MQSCO
- PMQSD – Pointer to a data structure of type MQSD
- PMQSMPO – Pointer to a data structure of type MQSMPO
- PMQSRO – Pointer to a data structure of type MQSRO
- PMQSTS – Pointer to a data structure of type MQSTS
- PMQTID – Pointer to a thread ID
- PMQTM – Pointer to a data structure of type MQTM
- PMQTM2 – Pointer to a data structure of type MQTM2
- PMQUINT8 – Pointer to a data type of MQUINT8
- PMQUINT16 – Pointer to a data type of MQUINT16
- PMQUINT32 – Pointer to a data type of MQUINT32
- PMQUINT64 – Pointer to a data type of MQUINT64
- PMQULONG – Pointer to a data type of MQULONG
- PMQVOID – Pointer
- PMQWIH – Pointer to a data structure of type MQWIH
- PMQXQH – Pointer to a data structure of type MQXQH

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

MQBOOL - boolean

The MQBOOL data type represents a boolean value. The value 0 represents false. Any other value represents true.

An MQBOOL must be aligned as for the MQLONG data type.

MQBYTE - byte

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

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

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

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

MQBYTE n - string of n bytes

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

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

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

Named constants are available to define the lengths of byte string fields. These are listed in *WebSphere MQ Constants*.

MQCHAR - character

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

When MQCHAR data is sent between queue managers that use different character sets or encodings, the MQCHAR data usually requires conversion in order for the data to be interpreted correctly. The queue manager does this automatically for MQCHAR data in the MQMD structure. Conversion of MQCHAR data in the application message data is controlled by the MQGMO_CONVERT option specified on the MQGET call; see the description of this option in “MQGMO – Get-message options” on page 126 for further details.

MQCHAR n - string of n characters

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

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

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

Named constants are available that define the lengths of character string fields and are listed in *WebSphere MQ Constants*.

MQFLOAT32 - 32-bit floating-point number

The MQFLOAT32 data type is a 32-bit floating-point number represented using the standard IEEE floating-point format. An MQFLOAT32 must be aligned on a 4-byte boundary.

The use of MQFLOAT32 in C on z/OS® requires the use of the FLOAT(IEEE) compiler flag.

The use of MQFLOAT32 in COBOL is limited to compilers that support floating-point numbers in IEEE format. This may require the use of the FLOAT(NATIVE) compiler flag.

MQFLOAT64 - 64-bit floating-point number

The MQFLOAT64 data type is a 64-bit floating-point number represented using the standard IEEE floating-point format. An MQFLOAT64 must be aligned on a 8-byte boundary.

The use of MQFLOAT64 in C on z/OS requires the use of the FLOAT(IEEE) compiler flag.

The use of MQFLOAT64 in COBOL is limited to compilers that support floating-point numbers in IEEE format. This may require the use of the FLOAT(NATIVE) compiler flag.

MQHCONFIG - configuration handle

The MQHCONFIG data type represents a configuration handle, that is, the component that is being configured for a particular installable service. A configuration handle must be aligned on its natural boundary.

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

MQHCONN - connection handle

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

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

MQHMSG - message handle

The MQHMSG data type represents a message handle that gives access to a message. A message handle must be aligned on an 8-byte boundary.

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

MQHOBJ - object handle

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

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

MQINT8 - 8-bit signed integer

The MQINT8 data type is an 8-bit signed integer that can take any value in the range -128 to +127, unless otherwise restricted by the context.

MQINT16 - 16-bit signed integer

The MQINT16 data type is a 16-bit signed integer that can take any value in the range -32 768 to +32 767, unless otherwise restricted by the context. An MQINT16 must be aligned on a 2-byte boundary.

MQINT32 - 32-bit signed integer

The MQINT32 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.

See the “MQLONG” on page 6 definition.

MQINT64

The MQINT64 data type is a 64-bit signed integer that can take any value in the range -9 223 372 036 854 775 808 through +9 223 372 036 854 775 807, unless otherwise restricted by the context.

For COBOL, the valid range is limited to -999 999 999 999 999 999 through +999 999 999 999 999 999. A 64-bit integer must be aligned on an 8-byte boundary.

MQLONG

The MQLONG data type is a 32-bit signed binary integer that can take any value in the range -2 147 483 648 through +2 147 483 647, unless otherwise restricted by the context.

For COBOL, the valid range is limited to -999 999 999 through +999 999 999. An MQLONG must be aligned on a 4-byte boundary.

MQPID - process identifier

The WebSphere MQ process identifier.

This is the same identifier used in MQ trace and FFST™ dumps, but might be different from the operating system process identifier.

MQPTR - pointer

The MQPTR data type is the address of data of any type. A pointer must be aligned on its natural boundary; this is a 16-byte boundary on i5/OS®, and an 8-byte boundary on other platforms.

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

MQTID - thread identifier

The WebSphere MQ thread identifier.

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

MQUINT8 - unsigned 8-bit integer

The MQUINT8 data type is an 8-bit unsigned integer that can take any value in the range 0 to +255, unless otherwise restricted by the context.

MQUINT16 - unsigned 16-bit integer

The MQUINT16 data type is a 16-bit unsigned integer that can take any value in the range 0 through +65 535, unless otherwise restricted by the context. An MQUINT16 must be aligned on a 2-byte boundary.

MQUINT32 - 32-bit unsigned integer

The MQUINT32 data type is a 32-bit unsigned binary integer.

See the definition of “MQULONG - 32-bit unsigned integer” on page 7 MQULONG.

MQUINT64 - 64-bit unsigned integer

The MQUINT64 data type is a 64-bit unsigned integer that can take any value in the range 0 through +18 446 744 073 709 551 615, unless otherwise restricted by the context.

For COBOL, the valid range is limited to 0 through +999 999 999 999 999. A 64-bit integer must be aligned on an 8-byte boundary.

MQULONG - 32-bit unsigned integer

The MQULONG data type is a 32-bit unsigned binary integer that can take any value in the range 0 through +4 294 967 294, unless otherwise restricted by the context.

For COBOL, the valid range is limited to 0 through +999 999 999. An MQULONG must be aligned on a 4-byte boundary.

PMQACH - pointer to a data structure of type MQACH

A pointer to a data structure of type MQACH.

PMQAIR - pointer to a data structure of type MQAIR

A pointer to a data structure of type MQAIR.

PMQAXC - pointer to a data structure of type MQAXC

A pointer to a data structure of type MQAXC.

PMQAXP - pointer to a data structure of type MQAXP

A pointer to a data structure of type MQAXP.

PMQBMHO - pointer to a data structure of type MQBMHO

A pointer to a data structure of type MQBMHO.

PMQBO - pointer to a data structure of type MQBO

A pointer to a data structure of type MQBO.

PMQBOOL - pointer to data of type MQBOOL

A pointer to data of type MQBOOL.

PMQBYTE - pointer to a data type of MQBYTE

A pointer to a data type of MQBYTE.

PMQBYTE_n - pointer to a data structure of type MQBYTE_n

A pointer to a data structure of type MQBYTE_n, where n can be 8, 12, 16, 24, 32, 40, 48 or 128.

PMQCBC - pointer to a data structure of type MQCBC

A pointer to a data structure of type MQCBC.

PMQCBD - pointer to a data structure of type MQCBD

A pointer to a data structure of type MQCBD.

PMQCHAR - pointer to data of type MQCHAR

A pointer to data of type MQCHAR.

PMQCHAR_n - pointer to a data type of MQCHAR_n

A pointer to a data type of MQCHAR_n, where n can be 4, 8, 12, 20, 28, 32, 64, 128, 256, 264.

| **PMQCHARV - pointer to a data structure of type MQCHARV**

| A pointer to a data structure of type MQCHARV.

| **PMQCIH - pointer to a data structure of type of MQCIH**

| A pointer to a data structure of type of MQCIH

| .

| **PMQCMHO - pointer to a data structure of type MQCMHO**

| A pointer to a data structure of type MQCMHO.

| **PMQCNO - pointer to a data structure of type of MQCNO**

| A pointer to a data structure of type of MQCNO

| .

| **PMQCSP - pointer to a data structure of type MQCSP**

| A pointer to a data structure of type MQCSP.

| **PMQCTLO - pointer to a data structure of type MQCTLO**

| A pointer to a data structure of type MQCTLO.

| **PMQDH - pointer to a data structure of type MQDH**

| A pointer to a data structure of type MQDH.

| **PMQDHO - pointer to a data structure of type MQDHO**

| A pointer to a data structure of type MQDHO.

| **PMQDLH - pointer to a data structure of type of MQDLH**

| A pointer to a data structure of type of MQDLH.

| **PMQDMHO - pointer to a data structure of type MQDMHO**

| A pointer to a data structure of type MQDMHO.

| **PMQDMPO - pointer to a data structure of type MQDMPO**

| A pointer to a data structure of type MQDMPO.

| **PMQEPH - pointer to a data structure of type MQEPH**

| A pointer to a data structure of type MQEPH.

| **PMQFLOAT32 - pointer to data of type MQFLOAT32**

| A pointer to data of type MQFLOAT32.

| **PMQFLOAT64 - pointer to data of type MQFLOAT64**

| A pointer to data of type MQFLOAT64.

| **PMQFUNC - pointer to a function**

| A pointer to a function.

| **PMQGMO - pointer to a data structure of type MQGMO**

| A pointer to a data structure of type MQGMO.

| **PMQHCONFIG - pointer to a data type of MQHCONFIG**

| A pointer to a data type of MQHCONFIG.

| **PMQHCONN - pointer to a data type of MQHCONN**

| A pointer to a data type of MQHCONN.

| **PMQHMSG - pointer to a data type of MQHMSG**

| A pointer to a data type of MQHMSG.

| **PMQHOBJ - pointer to data of type MQHOBJ**

| A pointer to data of type MQSMPO.

| **PMQIIH - pointer to a data structure of type MQIIH**

| A pointer to a data structure of type MQIIH.

| **PMQIMPO - pointer to a data structure of type MQIMPO**

| A pointer to a data structure of type MQIMPO.

| **PMQINT8 - pointer to data of type MQINT8**

| A pointer to data of type MQINT8.

| **PMQINT16 - pointer to data of type MQINT16**

| A pointer to data of type MQINT16.

| **PMQINT32 - pointer to data of type MQINT32**

| A pointer to data of type MQINT32.

| **PMQINT64 - pointer to data of type MQINT64**

| A pointer to data of type MQINT64.

| **PMQLONG - pointer to data of type MQLONG**

| A pointer to data of type MQLONG.

| **PMQMD - pointer to structure of type MQMD**

| A pointer to structure of type MQMD.

| **PMQMDE - pointer to a data structure of type MQMDE**

| A pointer to a data structure of type MQMDE.

| **PMQMD1 - pointer to a data structure of type MQMD1**

| A pointer to a data structure of type MQMD1

| **PMQMD2 - pointer to a data structure of type MQMD2**

| A pointer to a data structure of type MQMD2

| **PMQMHBO - pointer to a data structure of type MQMHBO**

| A pointer to a data structure of type MQMHBO.

| **PMQOD - pointer to a data structure of type MQOD**

| A pointer to a data structure of type MQOD.

| **PMQOR - pointer to a data structure of type MQOR**

| A pointer to a data structure of type MQOR.

| **PMQPD - pointer to a data structure of type MQPD**

| A pointer to a data structure of type MQPD.

| **PMQPID - pointer to a process identifier**

| A pointer to a process identifier.

| **PMQMD - pointer to a data structure of type MQMD**

| A pointer to a data structure of type MQMD.

| **PMQPMO - pointer to a data structure of type MQPMO**

| A pointer to a data structure of type MQPMO.

| **PMQPTR - pointer to data of type MQPTR**

| A pointer to data of type MQPTR.

| **PMQRFH - pointer to a data structure of type MQRFH**

| A pointer to a data structure of type MQRFH.

| **PMQRFH2 - pointer to a data structure of type MQRFH2**

| A pointer to a data structure of type MQRFH2.

| **PMQRMH - pointer to a data structure of type MQRMH**

| **PMQRR - pointer to a data structure of type MQRR**

| A pointer to a data structure of type MQRR.

| **PMQSCO - pointer to a data structure of type MQSCO**

| A pointer to a data structure of type MQSCO.

| .

| **PMQSD - pointer to a data structure of type MQSD**

| A pointer to a data structure of type MQSD.

| **PMQSMPO - pointer to a data structure of type MQSMPO**

| A pointer to a data structure of type MQSMPO.

| **PMQSRO - pointer to a data structure of type MQSRO**

| A pointer to a data structure of type MQSRO.

| **PMQSTS - pointer to a data structure of type MQSTS**

| A pointer to a data structure of type MQSTS.

| **PMQTID - pointer to a data structure of type MQTID**

| A pointer to a data structure of type MQTID.

| **PMQTMA - pointer to a data structure of type MQTM**

| A pointer to a data structure of type MQTM.

| **PMQTMC2 - pointer to a data structure of type MQTMC2**

| **PMQUINT8 - pointer to data of type MQUINT8**

| A pointer to data of type MQUINT8.

| **PMQUINT16 - pointer to data of type MQUINT16**

| A pointer to data of type MQUINT16.

| **PMQUINT32 - pointer to data of type MQUINT32**

| A pointer to data of type MQUINT32.

| **PMQUINT64 - pointer to data of type MQUINT64**

| A pointer to data of type MQUINT64.

PMQULONG - pointer to data of type MQULONG

A pointer to data of type MQULONG.

PMQVOID - pointer

The PMQVOID data type is a pointer.

PMQWIH - pointer to a data structure of type MQWIH

A pointer to a data structure of type MQWIH.

PMQXQH - pointer to a data structure of type MQXQH

A pointer to a data structure of type MQXQH.

C declarations

Data type	Representation
MQBOOL	typedef MQLONG MQBOOL;
MQBYTE	typedef unsigned char MQBYTE;
MQBYTE8	typedef MQBYTE MQBYTE8[8];
MQBYTE16	typedef MQBYTE MQBYTE16[16];
MQBYTE24	typedef MQBYTE MQBYTE24[24];
MQBYTE32	typedef MQBYTE MQBYTE32[32];
MQBYTE40	typedef MQBYTE MQBYTE40[40];
MQCHAR	typedef char MQCHAR;
MQCHAR4	typedef MQCHAR MQCHAR4[4];
MQCHAR8	typedef MQCHAR MQCHAR8[8];
MQCHAR12	typedef MQCHAR MQCHAR12[12];
MQCHAR20	typedef MQCHAR MQCHAR20[20];
MQCHAR28	typedef MQCHAR MQCHAR28[28];
MQCHAR32	typedef MQCHAR MQCHAR32[32];
MQCHAR48	typedef MQCHAR MQCHAR48[48];
MQCHAR64	typedef MQCHAR MQCHAR64[64];
MQCHAR128	typedef MQCHAR MQCHAR128[128];
MQCHAR256	typedef MQCHAR MQCHAR256[256];
MQFLOAT32	typedef float MQFLOAT32;
MQFLOAT64	typedef double MQFLOAT64;
MQHCONFIG	typedef void MQPOINTER MQHCONFIG;
MQHCONN	typedef MQLONG MQHCONN;
MQHOBJ	typedef MQLONG MQHOBJ;
MQINT8	typedef signed char MQINT8;
MQINT16	typedef short MQINT16;

Data type	Representation
MQINT64	<p>On 64-bit UNIX[®] systems: typedef long;</p> <p>On 32-bit AIX[®], Solaris, and HP-UX: typedef int64_t;</p> <p>On i5/OS, Linux[®], and z/OS: typedef long long;</p> <p>On Windows[®]: typedef _int64;</p>
MQLONG	<p>On i5/OS: typedef long MQLONG;</p> <p>other platforms: if defined(MQ_64_BIT) typedef int MQLONG; else typedef long MQLONG;</p>
MQPID	typedef MQLONG MQPID;
MQPTR	typedef void MQPOINTER MQPTR;
MQTID	typedef MQLONG MQTID;
MQUINT8	typedef unsigned char MQUINT8;
MQUINT16	typedef unsigned short MQUINT16;
MQUINT64	<p>On 64-bit UNIX systems: typedef unsigned long;</p> <p>On 32-bit AIX, Solaris, and HP-UX: typedef uint64_t;</p> <p>On i5/OS, Linux, and z/OS: typedef unsigned long long;</p> <p>On Windows: typedef unsigned _int64;</p>
MQULONG	<p>On i5/OS: typedef unsigned long MQULONG;</p> <p>other platforms: if defined(MQ_64_BIT) typedef unsigned int MQULONG; else typedef unsigned long MQULONG;</p>
PMQBO	typedef MQBO MQPOINTER PMQBO;
PMQBOOL	typedef MQBOOL MQPOINTER PMQBOOL;
PMQBYTE	typedef MQBYTE MQPOINTER PMQBYTE;
PMQBYTE8	typedef MQBYTE8[8] MQPOINTER PMQBYTE8[8];
PMQBYTE16	typedef MQBYTE16[16] MQPOINTER PMQBYTE16[16];
PMQBYTE24	typedef MQBYTE24[24] MQPOINTER PMQBYTE24[24];

Data type	Representation
PMQBYTE32	typedef MQBYTE32[32] MQPOINTER PMQBYTE32[32];
PMQBYTE40	typedef MQBYTE40[40] MQPOINTER PMQBYTE40[40];
PMQBYTE128	typedef MQBYTE128[128] MQPOINTER PMQBYTE128[128];
PMQCHAR	typedef MQCHAR MQPOINTER PMQCHAR;
PMQCHAR4	typedef MQCHAR4[4] MQPOINTER PMQCHAR4[4];
PMQCHAR8	typedef MQCHAR8[8] MQPOINTER PMQCHAR8[8];
PMQCHAR12	typedef MQCHAR12[12] MQPOINTER PMQCHAR12[12];
PMQCHAR20	typedef MQCHAR20[20] MQPOINTER PMQCHAR20[20];
PMQCHAR28	typedef MQCHAR28[28] MQPOINTER PMQCHAR28[28];
PMQCHAR32	typedef MQCHAR32[32] MQPOINTER PMQCHAR32[32];
PMQCHAR48	typedef MQCHAR48[48] MQPOINTER PMQCHAR48[48];
PMQCHAR64	typedef MQCHAR64[64] MQPOINTER PMQCHAR64[64];
PMQCHAR128	typedef MQCHAR128[128] MQPOINTER PMQCHAR128[128];
PMQCHAR256	typedef MQCHAR256[256] MQPOINTER PMQCHAR256[256];
PMQCHAR264	typedef MQCHAR264[264] MQPOINTER PMQCHAR264[264];
PMQCIH	typedef MQCIH MQPOINTER PMQCIH;
PMQCNO	typedef MQCNO MQPOINTER PMQCNO;
PMQDLH	typedef MQDLH MQPOINTER PMQDLH;
PMQFUNC	typedef void MQPOINTER PMQFUNC;
PMQFLOAT32	typedef MQFLOAT32 MQPOINTER PMQFLOAT32;
PMQFLOAT64	typedef MQFLOAT64 MQPOINTER PMQFLOAT64;
PMQGMO	typedef MQGMO MQPOINTER PMQGMO;
PMQHCONFIG	typedef MQHCONFIG MQPOINTER PMQHCONFIG;
PMQHCONN	typedef MQHCONN MQPOINTER PMQHCONN;
PMQHOBJ	typedef MQHOBJ MQPOINTER PMQHOBJ;
PMQIIH	typedef MQIIH MQPOINTER PMQIIH;
PMQINT8	typedef MQINT8 MQPOINTER PMQINT8;
PMQINT16	typedef MQINT16 MQPOINTER PMQINT16;
PMQLONG	typedef MQLONG MQPOINTER PMQLONG;
PMQMD	typedef MQMD MQPOINTER PMQMD;
PMQMD1	typedef MQMD1[1] MQPOINTER PMQMD1[1];
PMQMDE	typedef MQMDE MQPOINTER PMQMDE;
PMQOD	typedef MQOD MQPOINTER PMQOD;
PMQPMO	typedef MQPMO MQPOINTER PMQPMO;
PMQPTR	typedef MQPTR MQPOINTER PMQPTR;
PMQRFH	typedef MQRFH MQPOINTER PMQRFH;
PMQRFH2	typedef MQRFH2[2] MQPOINTER PMQRFH2[2];
PMQRMH	typedef MQRMH MQPOINTER PMQRMH;
PMQTM	typedef MQTM MQPOINTER PMQTM;
PMQTM2	typedef MQTM2[2] MQPOINTER PMQTM2[2];
PMQUINT8	typedef MQUINT8 MQPOINTER PMQUINT8;

Data type	Representation
PMQUINT16	typedef MQUINT16 MQPOINTER PMQUINT16;
PMQULONG	typedef MQULONG MQPOINTER PMQULONG;
PMQVOID	typedef void MQPOINTER PMQVOID;
PMQWIH	typedef MQWIH MQPOINTER PMQWIH;
PMQXQH	typedef MQXQH MQPOINTER PMQXQH;
PPMQBO	typedef PMQBO MQPOINTER PPMQBO;
PPMQBYTE	typedef PMQBYTE MQPOINTER PPMQBYTE;
PPMQCHAR	typedef PMQCHAR MQPOINTER PPMQCHAR;
PPMQCNO	typedef PMQCNO MQPOINTER PPMQCNO;
PPMQGMO	typedef PMQGMO MQPOINTER PPMQGMO;
PPMQHCONN	typedef PMQHCONN MQPOINTER PPMQHCONN;
PPMQHOBJ	typedef PMQHOBJ MQPOINTER PPMQHOBJ;
PPMQLONG	typedef PMQLONG MQPOINTER PPMQLONG;
PPMQMD	typedef PMQMD MQPOINTER PPMQMD;
PPMQOD	typedef PMQOD MQPOINTER PPMQOD;
PPMQPMO	typedef PMQPMO MQPOINTER PPMQPMO;
PPMQULONG	typedef PMQULONG MQPOINTER PPMQULONG;
PPMQVOID	typedef PMQVOID MQPOINTER PPMQVOID;

Where defined (MQ_64_BIT) means a 64 bit platform.

See “Data types” on page 21 for a description of the MQPOINTER macro variable.

COBOL declarations

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

Data type	Representation
MQCHAR256	PIC X(256)
MQFLOAT32	USAGE COMP-1
MQFLOAT64	USAGE COMP-2
MQHCONN	PIC S9(9) BINARY
MQHOBJ	PIC S9(9) BINARY
MQINT8	PIC S9(2) BINARY
MQINT16	PIC S9(4) BINARY
MQINT64	PIC S9(18) BINARY
MQLONG	PIC S9(9) BINARY
MQPTR	POINTER
MQUINT8	PIC 9(2) BINARY
MQUINT16	PIC 9(4) BINARY
MQUINT64	PIC 9(18) BINARY
MQULONG	PIC 9(9) BINARY

PL/I declarations

PL/I is supported on z/OS.

Data type	Representation
MQBOOL	fixed bin(31)
MQBYTE	char(1)
MQBYTE8	char(8)
MQBYTE16	char(16)
MQBYTE24	char(24)
MQBYTE32	char(32)
MQBYTE40	char(40)
MQCHAR	char(1)
MQCHAR4	char(4)
MQCHAR8	char(8)
MQCHAR12	char(12)
MQCHAR20	char(20)
MQCHAR28	char(28)
MQCHAR32	char(32)
MQCHAR48	char(48)
MQCHAR64	char(64)
MQCHAR128	char(128)
MQCHAR256	char(256)
MQFLOAT32	binary float(21) ieee
MQFLOAT64	binary float(52) ieee
MQHCONN	fixed bin(31)
MQHOBJ	fixed bin(31)

Data type	Representation
MQINT8	fixed bin(7)
MQINT16	fixed bin(15)
MQINT64	fixed bin(63)
MQLONG	fixed bin(31)
MQPTR	pointer
MQUINT8	fixed bin(8)
MQUINT16	fixed bin(16)
MQUINT64	fixed bin(64)
MQULONG	fixed bin(32)

System/390 assembler declarations

System/390[®] assembler is supported on z/OS only.

Data type	Representation
MQBOOL	DS F
MQBYTE	DS XL1
MQBYTE8	DS XL8
MQBYTE16	DS XL16
MQBYTE24	DS XL24
MQBYTE32	DS XL32
MQBYTE40	DS XL40
MQCHAR	DS CL1
MQCHAR4	DS CL4
MQCHAR8	DS CL8
MQCHAR12	DS CL12
MQCHAR20	DS CL20
MQCHAR28	DS CL28
MQCHAR32	DS CL32
MQCHAR48	DS CL48
MQCHAR64	DS CL64
MQCHAR128	DS CL128
MQCHAR256	DS CL256
MQFLOAT32	DS EB
MQFLOAT64	DS DB
MQHCONN	DS F
MQHOBJ	DS F
MQINT8	DS XL1
MQINT16	DS H
MQINT64	DS D
MQLONG	DS F
MQPTR	DS F

Data type	Representation
MQUINT8	DS XL1
MQUINT16	DS H
MQUINT64	DS D
MQULONG	DS F

Structure data types – introduction

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

Summary

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

Table 1. Structure data types used on MQI calls (or exit functions):

Structure	Description	Calls where used
MQACH	API exit chain header	
MQAIR	Authentication information record	MQCONN
MQAXC	API exit context	
MQAXP	API exit parameter	
MQBMHO	Buffer to message handle options	MQBUFMH
MQBO	Begin options	MQBEGIN
MQCBD	Callback descriptor	MQCB
MQCBO	Create-bag options	mqCreateBag
MQCHARV	Variable length string	MQINQMP
MQCNO	Connect options	MQCONN
MQCSP	Security parameters	MQCONN
MQCTLO	Callback options	MQCTL
MQDMPO	Delete message property options	MQDLTMP
MQGMO	Get-message options	MQGET
MQIMPO	Inquire message property options	MQINQMP
MQMD	Message descriptor	MQBUFMH, MQMHBUF, MQCB, MQGET, MQPUT, MQPUT1
MQMHBO	Message handle to buffer options	MQMHBUF
MQOD	Object descriptor	MQOPEN, MQPUT1
MQOR	Object record	MQOPEN, MQPUT1
MQPD	Property descriptor	MQSETMP
MQPMO	Put-message options	MQPUT, MQPUT1
MQPMR	Put-message record	MQPUT, MQPUT1
MQRR	Response record	MQOPEN, MQPUT, MQPUT1
MQSCO	SSL configuration options	MQCONN

Table 1. Structure data types used on MQI calls (or exit functions): (continued)

Structure	Description	Calls where used
MQSD	Subscription descriptor	MQSUB
MQSMPO	Set message property option	MQSETMP
MQSRO	Subscription request options	MQSUBRQ
MQSTS	Status reporting structure	MQSTAT

Table 2. Structure data types used in message data:

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

Note: The MQDXP structure (data conversion exit parameter) is described in Chapter 8, “Data conversion,” on page 677, together with the associated data conversion calls.

Rules for structure data types

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

- Structures must be aligned on their natural boundaries.
 - Most MQI structures require 4-byte alignment.
 - On i5/OS, structures containing pointers require 16-byte alignment; these are: MQCNO, MQOD, MQPMO.
- Each field in a structure must be aligned on its natural boundary.
 - Fields with data types that equate to MQLONG must be aligned on 4-byte boundaries.
 - Fields with data types that equate to MQPTR must be aligned on 16-byte boundaries on i5/OS, and 4-byte boundaries in other environments.
 - Other fields are aligned on 1-byte boundaries.
- The length of a structure must be a multiple of its boundary alignment.
 - Most MQI structures have lengths that are multiples of 4 bytes.

- On i5/OS, structures containing pointers have lengths that are multiples of 16 bytes.
4. Where necessary, padding bytes or fields must be added to ensure compliance with the above rules.

Conventions used in the descriptions

The description of each structure data type includes:

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

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

Structure name

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

Overview

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

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

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

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

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

output

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

input/output

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

Initial values

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

C declaration

Typical declaration of the structure in C.

COBOL declaration

Typical declaration of the structure in COBOL.

PL/I declaration

Typical declaration of the structure in PL/I.

System/390 assembler declaration

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

Visual Basic declaration

Typical declaration of the structure in Visual Basic.

C programming

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

Header files

Header files are provided to help you write C application programs that use the MQI.

These header files are summarized in Table 3.

Table 3. C header files

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

To improve the portability of applications, code the name of the header file in lowercase on the **#include** preprocessor directive:

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

Functions

You do not need to specify all parameters that are passed by address every time you invoke a function.

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

Where a particular parameter is not required, use a null pointer as the parameter on the function invocation, in place of the address of the parameter data. Parameters for which this is possible are identified in the call descriptions.

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

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

Parameters with undefined data type

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

Parameters of this sort are shown in the C examples as arrays of MQBYTE. You can declare the parameters in this way, but it is usually more convenient to declare them as the particular structure that describes the layout of the data in the message. Declare the actual function parameter as a pointer-to-void, and specify the address of any sort of data as the parameter on the function invocation.

Data types

Define all data types using the C **typedef** statement. For each data type, also define the corresponding pointer data type. The name of the pointer data type is the name of the elementary or structure data type prefixed with the letter P to denote a pointer. Define the attributes of the pointer using the MQPOINTER macro variable; the value of this macro variable depends on the environment. The following illustrates how to declare pointer datatypes:

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

Manipulating binary strings

Declare strings of binary data as one of the MQBYTEn data types.

Whenever you copy, compare, or set fields of this type, use the C functions **memcpy**, **memcmp**, or **memset**; for example:

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

MQMD MyMsgDesc;

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

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

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

Manipulating character strings

When the queue manager returns character data to the application, the queue manager always pads the character data with blanks to the defined length of the field; the queue manager *does not* return null-terminated strings.

Therefore, when copying, comparing, or concatenating such strings, use the string functions **strncpy**, **strncmp**, or **strncat**.

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

Initial values for structures

The header files define various macro variables that you can use to provide initial values for the MQ structures when you declare instances of those structures.

These macro variables have names of the form MQxxx_DEFAULT, where MQxxx represents the name of the structure. They are used in the following way:

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

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

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

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

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

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

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

Use this form to initialize the field when you declare an instance of the structure with values different from those provided by the MQMD_DEFAULT macro variable. (This is not always necessary; in some environments you can use the string form of the value in both situations. However, the array form is recommended for declarations, because this is required for compatibility with the C++ programming language.)

Initial values for dynamic structures

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

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

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

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

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

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

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

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

Use from C++

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

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

/* rest of header file */

#ifdef __cplusplus
}
#endif
```

Notational conventions

This information shows how to invoke the functions and declare parameters.

In some cases, the parameters are arrays whose size is not fixed. For these, a lowercase *n* is used to represent a numeric constant. When you code the declaration for that parameter, replace the *n* with the numeric value required.

COBOL programming

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

COPY files

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

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

- Use the structures with initial values in the **WORKING-STORAGE SECTION** of a COBOL program; they are contained in COPY files with names suffixed with the letter **V** (for Values).
- Use the structures without initial values in the **LINKAGE SECTION** of a COBOL program; they are contained in COPY files with names suffixed with the letter **L** (for Linkage).

The COPY files are summarized in Table 4. Not all the files listed are available in all environments.

Table 4. COBOL COPY files

File (with initial values)	File (without initial values)	Contents
CMQAIRV	CMQAIRL	Authentication information record
CMQBOV	CMQBOL	Begin options structure
CMQCIHV	CMQCIHL	CICS information header structure
CMQCNV	CMQCNOL	Connect options structure
CMQDHV	CMQDHL	Distribution header structure
CMQDLHV	CMQDLHL	Dead letter header structure
CMQDXPV	CMQDXPL	Data conversion exit parameter structure
CMQGMV	CMQGMOL	Get message options structure
CMQIIHV	CMQIIHL	IMS information header structure
CMQMDV	CMQMDL	Message descriptor structure
CMQMDEV	CMQMDEL	Message descriptor extension structure
CMQMD1V	CMQMD1L	Message descriptor structure version 1
CMQODV	CMQODL	Object descriptor structure
CMQORV	CMQORL	Object record structure
CMQPMV	CMQPMOL	Put message options structure
CMQRFHV	CMQRFHL	Rules and formatting header structure
CMQRFH2V	CMQRFH2L	Rules and formatting header structure version 2
CMQRMHV	CMQRMHL	Reference message header structure

Table 4. COBOL COPY files (continued)

File (with initial values)	File (without initial values)	Contents
CMQRRV	CMQRRL	Response record structure
CMQSCOV	CMQSCOL	SSL configuration options
CMQTMV	CMQTML	Trigger message structure
CMQTMCV	CMQTMCL	Trigger message structure (character format)
CMQTM2V	CMQTM2L	Trigger message structure (character format) version 2
CMQWIHV	CMQWIHL	Work information header structure
CMQXQHV	CMQXQHL	Transmission queue header structure
CMQV	–	Named constants for main MQI
CMQXV	–	Named constants for data conversion exit

Structures

In the COPY file, each structure declaration begins with a level-10 item; this enables you to declare several instances of the structure by coding the level-01 declaration and then using the **COPY** statement to copy in the remainder of the structure declaration. To refer to the appropriate instance, use the **IN** keyword:

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

Align the structures on appropriate boundaries. If you use the **COPY** statement to include a structure following an item that is not the level-01 item, ensure that the structure begins at the appropriate offset from the start of the level-01 item. Most MQI structures require 4-byte alignment; the exceptions to this are MQCNO, MQOD, and MQPMO, which require 16-byte alignment on i5/OS.

In this book, the names of fields in structures are shown without a prefix. In COBOL, the field names are prefixed with the name of the structure followed by a hyphen. However, if the structure name ends with a numeric digit, indicating that the structure is a second or later version of the original structure, the numeric digit is *omitted* from the prefix. Field names in COBOL are shown in uppercase (although lowercase or mixed case can be used if required). For example, the field *MsgType* described in “MQMD – Message descriptor” on page 181 becomes MQMD-MSGTYPE in COBOL.

The V-suffix structures are declared with initial values for all the fields; you need to set only those fields where you want a value that is different from the supplied initial value.

Pointers

Some structures need to address optional data that might be discontinuous with the structure, such as the MQOR and MQRR records addressed by the MQOD structure.

To address this optional data, the structures contain fields that are declared with the pointer data type. However, COBOL does not support the pointer data type in all environments. Because of this, the optional data can also be addressed using fields that contain the offset of the data from the start of the structure.

If you want to port an application between environments, ascertain whether the pointer data type is available in all the intended environments. If it is not, the application must address the optional data using the offset fields instead of the pointer fields.

In those environments where pointers are not supported, declare the pointer fields as byte strings of the appropriate length, with the initial value being the all-null byte string. Do not alter this initial value if you are using the offset fields.

Named constants

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

Constants that have character-string values use the single-quote character as the string delimiter (`'`). In some environments you might need to specify an appropriate compiler option to cause the compiler to accept the single quote as the string delimiter in place of the double quote.

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

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

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

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

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

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

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

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

Notational conventions

The later sections in this book show how to invoke the calls and declare parameters. In some cases, the parameters are tables or character strings whose size is not fixed. For these, a lowercase *n* is used to represent a numeric constant. When you code the declaration for that parameter, replace the *n* with the numeric value required.

System/390 assembler programming

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

Macros

Various macros are provided to help you to write assembler application programs that use the MQI.

There are two macros for named constants, and one macro for each of the structures. These files are summarized in Table 5.

Table 5. Assembler macros

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

Structures

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

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

Specifying the name of the structure:

To declare more than one instance of a structure, the macro prefixes the name of each field in the structure with a user-specifiable string and an underscore.

The string used is the label specified on the invocation of the macro. If no label is specified, the name of the structure is used to construct the prefix:

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

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

Specifying the form of the structure:

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

DSECT=YES

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

DSECT=NO

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

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

Controlling the version of the structure:

By default, the macros always declare the most recent version of each structure.

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

DCLVER=CURRENT

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

DCLVER=SPECIFIED

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

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

The value specified must be uppercase. If you omit the **DCLVER** parameter, the value used is taken from the **MQDCLVER** global macro variable. You can set this variable using the **CMQVERA** macro (see below).

Declaring one structure embedded within another:

To declare one structure as a component of another structure, use the **NESTED** parameter:

NESTED=YES

The structure declaration is nested within another.

NESTED=NO

The structure declaration is not nested within another.

The value specified must be uppercase. If you omit the **NESTED** parameter, **NESTED=NO** is assumed.

Specifying initial values for fields:

Specify the value to be used to initialize a field in a structure by coding the name of that field (without the prefix) as a parameter on the macro invocation, accompanied by the value required.

For example, to declare a message-descriptor structure with the *MsgType* field initialized with **MQMT_REQUEST**, and the *ReplyToQ* field initialized with the string "MY_REPLY_TO_QUEUE", use the following:

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

If you specify a named constant (equate) as a value on the macro invocation, use the **CMQA** macro to define the named constant. Do not enclose character string values in single quotes.

Controlling the listing:

Control the appearance of the structure declaration in the assembler listing using the **LIST** parameter:

LIST=YES

The structure declaration appears in the assembler listing.

LIST=NO

The structure declaration does not appear in the assembler listing.

The value specified must be uppercase. If you omit the **LIST** parameter, **LIST=NO** is assumed.

CMQVERA macro

This macro allows you to set the default value to be used for the **DCLVER** parameter on the structure macros. The value specified by **CMQVERA** is used by the structure macro only if you omit the **DCLVER** parameter from the invocation of the structure macro. The default value is set by coding the **CMQVERA** macro with the **DCLVER** parameter:

DCLVER=CURRENT

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

DCLVER=SPECIFIED

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

You must specify the **DCLVER** parameter, and the value must be uppercase. The value set by **CMQVERA** remains the default value until the next invocation of **CMQVERA**, or the end of the assembly. If you omit **CMQVERA**, the default is **DCLVER=CURRENT**.

Notational conventions

The later sections in this book show how to invoke the calls and declare parameters. In some cases, the parameters are arrays or character strings whose size is not fixed. For these, a lowercase **n** is used to represent a numeric constant. When you code the declaration for that parameter, replace the **n** with the numeric value required.

MQAIR – Authentication information record

The following table summarizes the fields in the structure.

Table 6. Fields in MQAIR

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

Overview for MQAIR

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

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

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

Fields for MQAIR

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

AuthInfoConnName (MQCHAR264)

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

If the value is shorter than the length of the field, terminate the value with a null character, or pad it with blanks to the length of the field. If the value is not valid, the call fails with reason code MQRC_AUTH_INFO_CONN_NAME_ERROR.

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

AuthInfoType (MQLONG)

This is the type of authentication information contained in the record.

The value must be:

MQAIT_CRL_LDAP

Certificate revocation using LDAP server.

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

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

LDAPPassword (MQCHAR32)

This is the password needed to access the LDAP CRL server. If the value is shorter than the length of the field, terminate the value with a null character, or pad it with blanks to the length of the field.

If the LDAP server does not require a password, or you omit the LDAP user name, *LDAPPassword* must be null or blank. If you omit the LDAP user name and *LDAPPassword* is not null or blank, the call fails with reason code `MQRC_LDAP_PASSWORD_ERROR`.

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

LDAPUserNameLength (MQLONG)

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

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

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

LDAPUserNameOffset (MQLONG)

This is the offset in bytes of the LDAP user name from the start of the MQAIR structure.

The offset can be positive or negative. The field is ignored if *LDAPUserNameLength* is zero.

You can use either *LDAPUserNamePtr* or *LDAPUserNameOffset* to specify the LDAP user name, but not both; see the description of the *LDAPUserNamePtr* field for details.

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

LDAPUserNamePtr (PMQCHAR)

This is the LDAP user name.

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

You can supply the LDAP user name in one of two ways:

- By using the pointer field *LDAPUserNamePtr*
In this case, the application can declare a string that is separate from the MQAIR structure, and set *LDAPUserNamePtr* to the address of the string.
Using *LDAPUserNamePtr* is recommended for programming languages that support the pointer data type in a fashion that is portable to different environments (for example, the C programming language).
- By using the offset field *LDAPUserNameOffset*
In this case, the application must declare a compound structure containing the MQSCO structure followed by the array of MQAIR records followed by the LDAP user name strings, and set *LDAPUserNameOffset* to the offset of the appropriate name string from the start of the MQAIR structure. Ensure that this value is correct, and has a value that can be accommodated within an MQLONG (the most restrictive programming language is COBOL, for which the valid range is -999 999 999 through +999 999 999).
Using *LDAPUserNameOffset* is recommended for programming languages that do not support the pointer data type, or that implement the pointer data type in a fashion that might not be portable to different environments (for example, the COBOL programming language).

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

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

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

StrucId (MQCHAR4)

The value must be:

MQAIR_STRUC_ID

Identifier for the authentication information record.

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

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

Version (MQLONG)

The value must be:

MQAIR_VERSION_1

Version-1 authentication information record.

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

MQAIR_CURRENT_VERSION

Current version of authentication information record.

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

Initial values and language declarations for MQAIR

Table 7. Initial values of fields in MQAIR for MQAIR

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQAIR_STRUC_ID	'AIRb'
<i>Version</i>	MQAIR_CURRENT_VERSION	1
<i>AuthInfoType</i>	MQAIT_CRL_LDAP	1
<i>AuthInfoConnName</i>	None	Null string or blanks
<i>LDAPUserNamePtr</i>	None	Null pointer or null bytes
<i>LDAPUserNameOffset</i>	None	0
<i>LDAPUserNameLength</i>	None	0
<i>LDAPPassword</i>	None	Null string or blanks
Notes:		
<ol style="list-style-type: none"> 1. The symbol <i>b</i> represents a single blank character. 2. In the C programming language, the macro variable MQAIR_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure: <pre>MQAIR MyAIR = {MQAIR_DEFAULT};</pre> 		

C declaration

```
typedef struct tagMQAIR MQAIR;
struct tagMQAIR {
    MQCHAR4    StrucId;           /* Structure identifier */
    MQLONG     Version;          /* Structure version number */
    MQLONG     AuthInfoType;     /* Type of authentication
                                information */
    MQCHAR264  AuthInfoConnName; /* Connection name of CRL LDAP
                                server */
    PMQCHAR    LDAPUserNamePtr;  /* Address of LDAP user name */
    MQLONG     LDAPUserNameOffset; /* Offset of LDAP user name from start
                                of MQAIR structure */
    MQLONG     LDAPUserNameLength; /* Length of LDAP user name */
    MQCHAR32   LDAPPassword;     /* Password to access LDAP server */
};
```

COBOL declaration

```
** MQAIR structure
10 MQAIR.
** Structure identifier
15 MQAIR-STRUCID          PIC X(4).
** Structure version number
15 MQAIR-VERSION         PIC S9(9) BINARY.
** Type of authentication information
15 MQAIR-AUTHINFOTYPE    PIC S9(9) BINARY.
** Connection name of CRL LDAP server
15 MQAIR-AUTHINFOCONNNAME PIC X(264).
```

```

**   Address of LDAP user name
    15 MQAIR-LDAPUSERNAMEPTR    POINTER.
**   Offset of LDAP user name from start of MQAIR structure
    15 MQAIR-LDAPUSERNAMEOFFSET PIC S9(9) BINARY.
**   Length of LDAP user name
    15 MQAIR-LDAPUSERNAMELENGTH PIC S9(9) BINARY.
**   Password to access LDAP server
    15 MQAIR-LDAPPASSWORD      PIC X(32).

```

PL/I declaration

```

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

```

Visual Basic declaration

```

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

```

MQBMHO – Buffer to message handle options

The following table summarizes the fields in the structure. MQBMHO structure - buffer to message handle options

Table 8. Fields in MQBMHO

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>Options</i>	Options controlling the action of MQBMHO	Options

Overview for MQBMHO

Availability: All. Buffer to message handle options structure - overview

Purpose: The MQBMHO structure allows applications to specify options that control how message handles are produced from buffers. The structure is an input parameter on the MQBUFMH call.

Character set and encoding: Data in MQBMHO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields for MQBMHO

Buffer to message handle options structure - fields

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

Options (MQLONG)

Buffer to message handle structure - Options field

The value can be:

MQBMHO_DELETE_PROPERTIES

Properties that are added to the message handle are deleted from the buffer. If the call fails no properties are deleted.

Default options: If you do not need the option described, use the following option:

MQBMHO_NONE

No options specified.

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

StrucId (MQCHAR4)

Buffer to message handle structure - StrucId field

This is the structure identifier. The value must be:

MQBMHO_STRUC_ID

Identifier for buffer to message handle structure.

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

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

Version (MQLONG)

Buffer to message handle structure - Version field

This is the structure version number. The value must be:

MQBMHO_VERSION_1

Version number for buffer to message handle structure.

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

MQBMHO_CURRENT_VERSION

Current version of buffer to message handle structure.

This is always an input field. The initial value of this field is MQBMHO_VERSION_1.

Initial values and language declarations for MQBMHO

Buffer to message handle structure - Initial values

Table 9. Initial values of fields in MQBMHO

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQBMHO_STRUC_ID	'BMHO'
<i>Version</i>	MQBMHO_VERSION_1	1
<i>Options</i>	MQBMHO_NONE	0
Notes:		
1. In the C programming language, the macro variable MQBMHO_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure: MQBMHO MyBMHO = {MQBMHO_DEFAULT};		

C declaration

Buffer to message handle structure - C language declaration

```
typedef struct tagMQBMHO MQBMHO;
struct tagMQBMHO {
    MQCHAR4  StrucId;      /* Structure identifier */
    MQLONG   Version;     /* Structure version number */
    MQLONG   Options;     /* Options that control the action of
                          MQBUFMH */
};
```

COBOL declaration

Buffer to message handle structure - COBOL language declaration

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

PL/I declaration

Buffer to message handle structure - PL/I language declaration

```
Dcl
  1 MQBMHO based,
  3 StrucId      char(4),      /* Structure identifier */
  3 Version      fixed bin(31), /* Structure version number */
  3 Options      fixed bin(31), /* Options that control the action
                               of MQBUFMH */
```

System/390 assembler declaration

Buffer to message handle structure - Assembler language declaration

```
MQBMHO          DSECT
MQBMHO_STRUCID  DS   CL4  Structure identifier
MQBMHO_VERSION  DS   F    Structure version number
MQBMHO_OPTIONS  DS   F    Options that control the
*                action of MQBUFMH
MQBMHO_LENGTH   EQU   *-MQBMHO
MQBMHO_AREA     DS   CL(MQBMHO_LENGTH)
```

MQBO – Begin options

The following table summarizes the fields in the structure.

Table 10. Fields in MQBO

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

Overview for MQBO

Availability: AIX, HP-UX, i5/OS, Solaris, Linux, Windows; not available for WebSphere MQ clients.

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

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

Fields for MQBO

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

Options (MQLONG)

The value must be:

MQBO_NONE

No options specified.

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

StrucId (MQCHAR4)

The value must be:

MQBO_STRUC_ID

Identifier for begin-options structure.

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

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

Version (MQLONG)

The value must be:

MQBO_VERSION_1

Version number for begin-options structure.

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

MQBO_CURRENT_VERSION

Current version of begin-options structure.

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

Initial values and language declarations for MQBO

Table 11. Initial values of fields in MQBO for MQBO

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

Notes:

1. The symbol *b* represents a single blank character.
2. In the C programming language, the macro variable MQBO_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

```
MQBO MyBO = {MQBO_DEFAULT};
```

C declaration

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

COBOL declaration

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

PL/I declaration

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

Visual Basic declaration

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

MQCBC – Callback context

The following table summarizes the fields in the structure. Structure describing the callback routine.

Table 12. Fields in MQCBC

Field	Description	Topic
<i>StrucID</i>	Structure identifier	StrucID
<i>Version</i>	Structure version number	Version
<i>CallType</i>	Why function has been called	CallType
<i>Hobj</i>	Object handle	Hobj
<i>CallbackArea</i>	Field for callback function to use	CallbackArea
<i>ConnectionArea</i>	Field for callback function to use	ConnectionArea
<i>CompCode</i>	Completion code	CompCode
<i>Reason</i>	Reason code	Reason
<i>State</i>	Indication of the state of the current consumer	State
<i>DataLength</i>	Message length	DataLength
<i>BufferLength</i>	Length of message buffer in bytes	BufferLength
<i>Flags</i>	General flags	Flags

Overview for MQCBC

Availability: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, z/OS, plus WebSphere MQ clients connected to these systems.

Purpose: The MQCBC structure is used to specify context information that is passed to a callback function.

The structure is an input/output parameter on the call to a message consumer routine.

Version: The current version of MQCBC is MQCBC_VERSION_1.

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

Fields for MQCBC

Alphabetic list of fields for the MQCBC structure.

The MQCBC structure contains the following fields; the fields are described in alphabetical order:

BufferLength (MQLONG)

The buffer can be larger than both the MaxMsgLength value defined for the consumer and the ReturnedLength value in the MQGMO. Callback context structure - BufferLength field

This is the length in bytes of the message buffer that has been passed to this function.

The actual message length is supplied in `DataLength` field.

The application can use the entire buffer for its own purposes for the duration of the callback function.

This is an input field to the message consumer function; it is not relevant to an exception handler function.

CallbackArea (MQPTR)

Callback context structure - `CallbackArea` field

This is a field that is available for the callback function to use.

The queue manager makes no decisions based on the contents of this field and it is passed unchanged from the `CallbackArea` field in the MQCBD structure, which is a parameter on the MQCB call used to define the callback function.

Changes to the *CallbackArea* are preserved across the invocations of the callback function for an *HObj*. This field is not shared with callback functions for other handles.

This is an input/output field to the callback function. The initial value of this field is a null pointer or null bytes.

CallType (MQLONG)

Callback Context structure - `CallType` field

Field containing information about why this function has been called; the following are defined.

Message delivery call types: These call types contain information about a message. The *DataLength* and *BufferLength* parameters are valid for these call types.

MQCBCT_MSG_REMOVED

The message consumer function has been invoked with a message that has been destructively removed from the object handle.

If the value of *CompCode* is `MQCC_WARNING`, the value of the *Reason* field is `MQRC_TRUNCATED_MSG_ACCEPTED` or one of the codes indicating a data conversion problem.

MQCBCT_MSG_NOT_REMOVED

The message consumer function has been invoked with a message that has not yet been destructively removed from the object handle. The message can be destructively removed from the object handle using the *MsgToken*.

The message might not have been removed because:

- The MQGMO options requested a browse operation, `MQGMO_BROWSE_*`
- The message is larger than the available buffer and the MQGMO options do not specify `MQGMO_ACCEPT_TRUNCATED_MSG`

If the value of *CompCode* is `MQCC_WARNING`, the value of the *Reason* field is `MQRC_TRUNCATED_MSG_FAILED` or one of the codes indicating a data conversion problem.

Callback control call types: These call types contain information about the control of the callback and do not contain details about a message. These call types are requested using Options in the MQCBD structure.

The *DataLength* and *BufferLength* parameters are not valid for these call types.

MQCBCT_REGISTER_CALL

The purpose of this call type is to allow the callback function to perform some initial setup.

The callback function is invoked immediately after the callback is registered, that is, upon return from an MQCB call using a value for the *Operation* field of MQOP_REGISTER.

This call type is used both for message consumers and event handlers.

If requested, this is the first invocation of the callback function.

The value of the *Reason* field is MQRC_NONE.

MQCBCT_START_CALL

The purpose of this call type is to allow the callback function to perform some setup when it is started, for example, reinstating resources that were cleaned up when it was previously stopped.

The callback function is invoked when the connection is started using either MQOP_START or MQOP_START_WAIT.

If a callback function is registered within another callback function, this call type is invoked when the callback returns.

This call type is used for message consumers only.

The value of the *Reason* field is MQRC_NONE.

MQCBCT_STOP_CALL

The purpose of this call type is to allow the callback function to perform some cleanup when it is stopped for a while, for example, cleaning up additional resources that have been acquired during the consuming of messages.

The callback function is invoked when an MQCTL call is issued using a value for the *Operation* field of MQOP_STOP.

This call type is used for message consumers only.

The value of the *Reason* field is set to indicate the reason for stopping.

MQCBCT_DEREGISTER_CALL

The purpose of this call type is to allow the callback function to perform final cleanup at the end of the consume process. The callback function is invoked when the:

- Callback function is deregistered using an MQCB call with MQOP_DEREGISTER.
- Queue is closed, causing an implicit deregister. In this instance the callback function is passed MQHO_UNUSABLE_HOBJ as the object handle.
- MQDISC call completes – causing an implicit close and, therefore, a deregister. In this case the connection is not disconnected immediately, and any ongoing transaction is not yet committed.

If any of these actions are taken inside the callback function itself, the action is invoked once the callback returns.

This call type is used both for message consumers and event handlers.
If requested, this is the last invocation of the callback function.
The value of the *Reason* field is set to indicate the reason for stopping.

MQCBCT_EVENT_CALL

Event handler function

The event handler function has been invoked without a message when the queue manager or connection stops or quiesces.:

This call can be used to take appropriate action for all callback functions.

- **Message consumer function**

The message consumer function has been invoked without a message when an error (*CompCode*= MQCC_FAILED) has been detected that is specific to the object handle; for example *Reason* code = MQRC_GET_INHIBITED.

The value of the *Reason* field is set to indicate the reason for the call.

This is an input field. MQCBCT_MSG_REMOVED and MQCBCT_MSG_NOT_REMOVED are applicable only to message consumer functions.

CompCode (MQLONG)

Callback context structure - CompCode field

This is the completion code. It indicates whether there were any problems consuming the message; it is one of the following:

MQCC_OK

Successful completion

MQCC_WARNING

Warning (partial completion)

MQCC_FAILED

Call failed

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

ConnectionArea (MQPTR)

Callback context structure - ConnectionArea field

This is a field that is available for the callback function to use.

The queue manager makes no decisions based on the contents of this field and it is passed unchanged from the ConnectionArea field in the MQCTLO structure, which is a parameter on the MQCTL call used to control the callback function.

Any changes made to this field by the callback functions are preserved across the invocations of the callback function. This area can be used to pass information that is to be shared by all callback functions. Unlike *CallbackArea*, this area is common across all callbacks for a connection handle.

This is an input and output field. The initial value of this field is a null pointer or null bytes.

DataLength (MQLONG)

This is the length in bytes of the application data in the message. If the value is zero, it means that the message contains no application data. Callback context structure - DataLength field

The DataLength field contains the length of the message but not necessarily the length of the message data passed to the consumer. It could be that the message was truncated. Use the ReturnedLength field in the MQGMO to determine how much data has actually been passed to the consumer.

If the reason code indicates the message has been truncated, you can use the DataLength field to determine how large the actual message is. This allows you to determine the size of the buffer required to accommodate the message data, and then issue an MQCB call to update the MaxMsgLength with an appropriate value.

If the MQGMO_CONVERT option is specified, the converted message could be larger than the value returned for DataLength. In such cases, the application probably needs to issue an MQCB call to update the MaxMsgLength to be greater than the value returned by the queue manager for DataLength.

To avoid message truncation problems, specify MaxMsgLength as MQCBD_FULL_MSG_LENGTH. This causes the queue manager to allocate a buffer for the full message length after data conversion. Be aware, however, that even if this option is specified, it is still possible that sufficient storage is not available to correctly process the request. Applications should always check the returned reason code. For example, if it is not possible to allocate sufficient storage to convert the message, the messages is returned to the application unconverted.

This is an input field to the message consumer function; it is not relevant to an event handler function.

Flags (MQLONG)

Flags containing information about this consumer. Callback context structure - Flags field

The following option is defined:

MQCBCF_READA_BUFFER_EMPTY

This flag can be returned if a previous MQCLOSE call using the MQCO_QUIESCE option failed with a reason code of MQRC_READ_AHEAD_MSGS.

This code indicated that the last read ahead message is being returned and that the buffer is now empty. If the application issues another MQCLOSE call using the MQCO_QUIESCE) option, it succeeds.

Note, that an application is not guaranteed to be given a message with this flag set, as there might still be messages in the read-ahead buffer that do not match the current selection criteria. In this instance, the consumer function is invoked with the reason code MQRC_HOBJ_QUIESCED.

If the read ahead buffer is completely empty, the consumer is invoked with the MQCBCF_READA_BUFFER_EMPTY flag and the reason code MQRC_HOBJ_QUIESCED_NO_MSGS.

This is an input field to the message consumer function; it is not relevant to an event handler function.

Hobj (MQHOBJ)

Callback context structure - Hobj field

This is the object handle for calls to the message consumer.

For an event handler, this value is MQHO_NONE

The application can use this handle and the message token in the Get Message Options block to get the message if a message has not been removed from the queue.

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

Reason (MQLONG)

Callback context structure - Reason field

This is the reason code qualifying the *CompCode*

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

State (MQLONG)

An indication as to the state of the current consumer. This field is of most value to an application when a nonzero reason code is passed to the consumer function. Callback context structure - State field

You can use this field to simplify application programming because you do not need to code behavior for each reason code.

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

State	Queue manager action	Value of constant
<i>MQCS_NONE</i> This reason code represents a normal call with no additional reason information	None; this is the normal operation.	0
<i>MQCS_SUSPENDED_TEMPORARY</i> These reason codes represent temporary conditions.	The callback routine is called to report the condition and then suspended. After a period of time the system might attempt the operation again, which can lead to the same condition being raised again.	1
<i>MQCS_SUSPENDED_USER_ACTION</i> These reason codes represent conditions where the callback needs to take action to resolve the condition.	The consumer is suspended and the callback routine is called to report the condition. The callback routine should resolve the condition if possible and either RESUME or close down the connection.	2
<i>MQCS_SUSPENDED</i> These reason codes represent failures that prevent further message callbacks.	The queue manager automatically suspends the callback function. If the callback function is resumed it is likely to receive the same reason code again.	3

State	Queue manager action	Value of constant
<i>MQCS_STOPPED</i> These reason codes represent the end of message consumption.	Delivered to the exception handler and to callbacks that specified MQCBDO_STOP_CALL. No further messages can be consumed.	4

StrucId (MQCHAR4)

Callback context structure - StrucId field

This is the structure identifier; the value must be:

MQCBC_STRUC_ID

Identifier for callback context structure.

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

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

Version (MQLONG)

Callback context structure - Version field

This is the structure version number; the value must be:

MQCBC_VERSION_1

Version-1 callback context structure.

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

MQCBC_CURRENT_VERSION

Current version of the callback context structure.

This is always an input field. The initial value of this field is MQCBC_VERSION_1.

The callback function is always passed the latest version of the structure.

Initial values and language declarations for MQCBC

Callback context structure - Initial values

Table 13. Initial values of fields in MQCBC

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQCBC_STRUC_ID	'CBCb'
<i>Version</i>	MQCBC_VERSION_1	1
<i>CallType</i>	None	0
<i>Hobj</i>	MQHO_UNUSABLE_HOBJ	-1
<i>CallbackArea</i>	None	Null pointer or null bytes
<i>ConnectionArea</i>	None	Null pointer or null bytes
<i>CompCode</i>	MQCC_OK	0
<i>Reason</i>	MQRC_NONE	0

Table 13. Initial values of fields in MQCBC (continued)

Field name	Name of constant	Value of constant
<i>State</i>	MQCS_NONE	0
<i>DataLength</i>	None	0
<i>BufferLength</i>	None	0
<i>Flags</i>	None	0
Notes: 1. The symbol <i>b</i> represents a single blank character. 2. In the C programming language, the macro variable MQCBC_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure: <pre>MQCBC MyCBC = {MQCBC_DEFAULT};</pre>		

C declaration

Callback context structure - C language declaration

```
typedef struct tagMQCBC MQCBC;
struct tagMQCBC {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQLONG    CallType;         /* Why Function was called */
    MQHOBJS   Hobj;            /* Object Handle */
    MQPTR     CallbackArea;     /* Callback data passed to the function */
    MQPTR     ConnectionArea;   /* MQCTL data area passed to the function */
    MQLONG    CompCode;         /* Completion Code */
    MQLONG    Reason;           /* Reason Code */
    MQLONG    State;            /* Consumer State */
    MQLONG    DataLength;       /* Message Data Length */
    MQLONG    BufferLength;      /* Buffer Length */
    MQLONG    Flags;            /* Flags containing information about
                                this consumer */
};
```

COBOL declaration

```
** MQCBC structure
 10 MQCBC.
** Structure Identifier
 15 MQCBC-STRUCID                PIC X(4).
** Structure Version
 15 MQCBC-VERSION                PIC S9(9) BINARY.
** Call Type
 15 MQCBC-CALLTYPE                PIC S9(9) BINARY.
** Object Handle
 15 MQCBC-HOBJ                    PIC S9(9) BINARY.
** Callback User Area
 15 MQCBC-CALLBACKAREA            POINTER
** Connection Area
 15 MQCBC-CONNECTIONAREA          POINTER
** Completion Code
 15 MQCBC-COMPCODE                PIC S9(9) BINARY.
** Reason Code
 15 MQCBC-REASON                  PIC S9(9) BINARY.
** Consumer State
 15 MQCBC-STATE                    PIC S9(9) BINARY.
** Data Length
 15 MQCBC-DATALENGTH              PIC S9(9) BINARY.
** Buffer Length
```

```

15 MQCBC-BUFFERLENGTH          PIC S9(9) BINARY.
** Flags
15 MQCBC-FLAGS                  PIC S9(9) BINARY.

```

PL/I declaration

```

dc1
1 MQCBC based,
3 StrucId          char(4),          /* Structure identifier */
3 Version          fixed bin(31),   /* Structure version */
3 CallType        fixed bin(31),   /* Callback type */
3 Hobj            fixed bin(31),   /* Object Handle */
3 CallbackArea    pointer,          /* User area passed to the function */
3 ConnectionArea pointer,          /* Connection User Area */
3 CompCode        fixed bin(31);   /* Completion Code */
3 Reason          fixed bin(31);   /* Reason Code */
3 State          fixed bin(31);   /* Consumer State */
3 DataLength      fixed bin(31);   /* Message Data Length */
3 BufferLength    fixed bin(31);   /* Message Buffer length */
3 Flags          fixed bin(31);   /* Consumer Flags */

```

MQCBD – Callback descriptor

The following table summarizes the fields in the structure. Structure specifying the callback function.

Table 14. Fields in MQCBD

Field	Description	Topic
<i>StrucID</i>	Structure identifier	StrucID
<i>Version</i>	Structure version number	Version
<i>CallbackType</i>	Type of callback function	CallbackType
<i>Options</i>	Options controlling message consumption	Options
<i>Callback Area</i>	Field for callback function to use	CallbackArea
<i>CallbackFunction</i>	Whether the function is invoked as an API call	CallbackFunction
<i>CallbackName</i>	Whether the function is invoked as a dynamically-linked program	CallbackName
<i>MaxMsgLength</i>	Length of longest message that can be read	MaxMsgLength

Overview for MQCBD

Availability: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, z/OS, and WebSphere MQ clients connected to these systems.

Purpose: The MQCBD structure is used to specify a callback function and the options controlling its use by the queue manager.

The structure is an input parameter on the MQCB call.

Version: The current version of MQCBD is MQCBD_VERSION_1.

Character set and encoding: Data in MQCBD must be in the character set and encoding of the local queue manager; these are given by the *CodedCharSetId*

queue-manager attribute and MQENC_NATIVE, respectively. However, if the application is running as an MQ client, the structure must be in the character set and encoding of the client.

Fields for MQCBD

Alphabetic list of fields for the MQCBD structure.

The MQCBD structure contains the following fields; the fields are described in alphabetical order:

CallbackArea (MQPTR)

Callback descriptor structure - CallbackArea field

This is a field that is available for the callback function to use.

The queue manager makes no decisions based on the contents of this field and it is passed unchanged from the CallbackArea field in the MQCBC structure, which is a parameter on the callback function declaration.

The value is used only on an *Operation* having a value MQOP_REGISTER, with no currently defined callback, it does not replace a previous definition.

This is an input and output field to the callback function. The initial value of this field is a null pointer or null bytes.

CallbackFunction (MQCB_FUNCTION)

Callback descriptor structure - CallbackFunction field

The callback function is invoked as a function call.

Use this field to specify a pointer to the call back function.

You *must* specify either *CallbackFunction* or *CallbackName*. If you specify both, the reason code MQRC_CALLBACK_ROUTINE_ERROR is returned.

If neither *CallbackName* nor *CallbackFunction* is not set, the call fails with the reason code MQRC_CALLBACK_ROUTINE_ERROR.

This option is not supported in the following environments:

- CICS on z/OS
- Programming languages and compilers that do not support function-pointer references

In such situations, the call fails with the reason code MQRC_CALLBACK_ROUTINE_ERROR.

On z/OS the function must expect to be called with OS linkage conventions. For example, in the C programming language, specify:

```
#pragma linkage(MQCB_FUNCTION,OS)
```

This is an input field. The initial value of this field is a null pointer or null bytes.

CallbackName (MQCHAR128)

Callback descriptor structure - CallbackName field

The call back function is invoked as a dynamically linked program.

You *must* specify either *CallbackFunction* or *CallbackName*. If you specify both, the reason code MQRC_CALLBACK_ROUTINE_ERROR is returned.

If neither *CallbackName* nor *CallbackFunction* is not set, the call fails with the reason code MQRC_CALLBACK_ROUTINE_ERROR.

The module is loaded when the first callback routine to use is registered, and unloaded when the last callback routine to use it deregisters.

Except where noted in the following text, the name is left-justified within the field, with no embedded blanks; the name itself is padded with blanks to the length of the field. In the descriptions that follow, square brackets ([]) denote optional information:

i5/OS The callback name can be one of the following formats:

- Library "/" Program
- Library "/" ServiceProgram ("FunctionName")

For example, MyLibrary/MyProgram(MyFunction).

The library name can be *LIBL. Both the library and program names are limited to a maximum of 10 characters.

UNIX systems

The callback name is the name of a dynamically-loadable module or library, suffixed with the name of a function residing in that library. The function name must be enclosed in parentheses. The library name can optionally be prefixed with a directory path:

[path]library(function)

If the path is not specified the system search path is used.

The name is limited to a maximum of 128 characters.

Windows

The callback name is the name of a dynamic-link library, suffixed with the name of a function residing in that library. The function name must be enclosed in parentheses. The library name can optionally be prefixed with a directory path and drive:

[d:][path]library(function)

If the drive and path are not specified the system search path is used.

The name is limited to a maximum of 128 characters.

z/OS The callback name is the name of a load module that is valid for specification on the EP parameter of the LINK or LOAD macro.

The name is limited to a maximum of 8 characters.

z/OS CICS

The callback name is the name of a load module that is valid for specification on the PROGRAM parameter of the EXEC CICS LINK command macro.

The name is limited to a maximum of 8 characters.

The program can be defined as remote using the REMOTESYSTEM option of the installed PROGRAM definition or by the dynamic routing program.

The remote CICS region must be connected to WebSphere MQ if the program is to use WebSphere MQ API calls. Note, however, that the *Hobj* field in the MQCBC structure is not valid in a remote system.

If a failure occurs trying to load *CallbackName*, one of the following error codes is returned to the application:

- MQRC_MODULE_NOT_FOUND
- MQRC_MODULE_INVALID
- MQRC_MODULE_ENTRY_NOT_FOUND

A message is also written to the error log containing the name of the module for which the load was attempted, and the failing reason code from the operating system.

This is an input field. The initial value of this field is a null string or blanks.

CallbackType (MQLONG)

Callback descriptor structure - CallbackType field

This is the type of the callback function. The value must be one of:

MQCBT_MESSAGE_CONSUMER

Defines this callback as a message consumer function.

A message consumer callback function is called when a message, meeting the selection criteria specified, is available on an object handle and the connection is started.

MQCBT_EVENT_HANDLER

Defines this callback as the asynchronous event routine; it is not driven to consume messages for a handle.

Hobj is not required on the MQCB call defining the event handler and is ignored if specified.

The event handler is called for conditions that affect the whole message consumer environment. The consumer function is invoked without a message when an event, for example, a queue manager or connection stopping, or quiescing, occurs. It is not called for conditions that are specific to a single message consumer, for example, MQRC_GET_INHIBITED.

Events are delivered to the application, regardless of whether the connection is started or stopped, except in the following environments:

- CICS on z/OS environment
- nonthreaded applications

If the caller does not pass one of these values, the call fails with a *Reason* code of MQRC_CALLBACK_TYPE_ERROR

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

MaxMsgLength (MQLONG)

This is the length in bytes of the longest message that can be read from the handle and given to the callback routine. Callback descriptor structure - MaxMsgLength field

If a message has a longer length, the callback routine receives *MaxMsgLength* bytes of the message, and reason code:

- MQRC_TRUNCATED_MSG_FAILED or
- MQRC_TRUNCATED_MSG_ACCEPTED if you specified MQGMO_ACCEPT_TRUNCATED_MSG.

The actual message length is supplied in the *DataLength* field of the MQCBC structure.

The following special value is defined:

MQCBD_FULL_MSG_LENGTH

The buffer length is adjusted by the system to return messages without truncation.

If insufficient memory is available to allocate a buffer to receive the message, the system calls the callback function with an MQRC_STORAGE_NOT_AVAILABLE reason code.

If, for example, you request data conversion, and there is insufficient memory available to convert the message data, the unconverted message is passed to the callback function.

This is an input field. The initial value of the *MaxMsgLength* field is MQCBD_FULL_MSG_LENGTH.

Options (MQLONG)

Callback descriptor structure - Options field

Any one, or all, of the following can be specified. If more than one option is required the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

MQCBDO_FAIL_IF QUIESCING

The MQCB call fails if the queue manager is in the quiescing state.

On z/OS, this option also forces the MQCB call to fail if the connection (for a CICS or IMS application) is in the quiescing state.

Specify MQGMO_FAIL_IF QUIESCING, in the MQGMO options passed on the MQCB call, to cause notification to message consumers when they are quiescing.

Control options: The following options control whether the callback function is called, without a message, when the state of the consumer changes:

MQCBDO_REGISTER_CALL

The callback function is invoked with call type MQCBCT_REGISTER_CALL.

MQCBDO_START_CALL

The callback function is invoked with call type MQCBCT_START_CALL.

MQCBDO_STOP_CALL

The callback function is invoked with call type MQCBCT_STOP_CALL.

MQCBDO_DEREGISTER_CALL

The callback function is invoked with call type MQCBCT_DEREGISTER_CALL.

See CallType for further details about these call types.

Default option: If you do not need any of the options described, use the following option:

MQCBDO_NONE

Use this value to indicate that no other options have been specified; all options assume their default values.

MQCBDO_NONE is defined to aid program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

This is an input field. The initial value of the *Options* field is MQCBDO_NONE.

StrucId (MQCHAR4)

Callback descriptor structure - StrucId field

This is the structure identifier; the value must be:

MQCBD_STRUC_ID

Identifier for callback descriptor structure.

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

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

Version (MQLONG)

Callback descriptor structure - Version field

This is the structure version number; the value must be:

MQCBD_VERSION_1

Version-1 callback descriptor structure.

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

MQCBD_CURRENT_VERSION

Current version of callback descriptor structure.

This is always an input field. The initial value of this field is MQCBD_VERSION_1.

Initial values and language declarations for MQCBD

Callback descriptor structure - Initial values

Table 15. Initial values of fields in MQCBD

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQCBD_STRUC_ID	'CBDb'
<i>Version</i>	MQCBD_VERSION_1	1
<i>CallbackType</i>	MQCBT_MESSAGE_CONSUMER	1
<i>Options</i>	MQCBDO_NONE	0
<i>CallbackArea</i>	None	Null pointer or null blanks

Table 15. Initial values of fields in MQCBD (continued)

Field name	Name of constant	Value of constant
<i>CallbackFunction</i>	None	Null pointer or null blanks
<i>CallbackName</i>	None	Null string or blanks
<i>MaxMsgLength</i>	MQCBD_FULL_MSG_LENGTH	-1
Notes: <ol style="list-style-type: none"> 1. The symbol <i>b</i> represents a single blank character. 2. The value Null string or blanks denotes the null sting in the C programming language, and blank characters in other programming languages. 3. In the C programming language, the macro variable MQCBD_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure: MQCBD MyCBD = {MQCBD_DEFAULT}; 		

C declaration

Callback descriptor structure - C language declaration

```
typedef struct tagMQCBD MQCBD;
struct tagMQCBD {
    MQCHAR4    StrucId;           /* Structure identifier */
    MQLONG     Version;          /* Structure version number */
    MQLONG     CallBackType;     /* Callback function type */
    MQLONG     Options;          /* Options controlling message
                                consumption */
    MQPTR      CallbackArea;     /* User data passed to the function */
    MQPTR      CallbackFunction; /* Callback function pointer */
    MQCHAR128  CallbackName;     /* Callback name */
    MQLONG     MaxMsgLength;     /* Maximum message length */
};
```

COBOL declaration

```
** MQCBCD structure
10  MQCBD.
** Structure Identifier
15  MQCBD-STRUCID                PIC X(4).
** Structure Version
15  MQCBD-VERSION                PIC S9(9) BINARY.
** Callback Type
15  MQCBD-CALLBACKTYPE           PIC S9(9) BINARY.
** Options
15  MQCBD-OPTIONS                PIC S9(9) BINARY.
** Callback User Area
15  MQCBD-CALLBACKAREA           POINTER
** Callback Function Pointer
15  MQCBD-CALLBACKFUNCTION        FUNCTION-POINTER
** Callback Program Name
15  MQCBD-CALLBACKNAME           PIC X(128)
** Maximum Message Length
15  MQCDB-MAXMSGLENGTH           PIC S9(9) BINARY.
```

PL/I declaration

```
dcl
1  MQCBD based,
3  StrucId          char(4),      /* Structure identifier*/
3  Version          fixed bin(31), /* Structure version*/
3  CallBackType     fixed bin(31), /* Callback function type */
3  Options          fixed bin(31), /* Options */
```



```

3 CallbackArea      pointer,      /* User area passed to the function */
3 CallbackFunction  pointer,      /* Callback Function Pointer */
3 CallbackName      char(128),    /* Callback Program Name */
3 MaxMsgLength      fixed bin(31); /* Maximum Message Length */

```

MQCHARV - Variable Length String

The following table summarizes the fields in the structure.

Field	Description	Topic
<i>VSPtr</i>	Pointer to the variable length string	VSPtr
<i>VSOffset</i>	Offset in bytes of the variable length string from the start of the structure that contains this MQCHARV structure	VSOffset
<i>VSLength</i>	The length in bytes of the variable length string addressed by the VSPtr or VSOffset field.	VSLength
<i>VSBufSize</i>	The size in bytes of the buffer addressed by the VSPtr or VSOffset field.	VSBufSize
<i>VSCCSID</i>	The character set identifier of the variable length string addressed by the VSPtr or VSOffset field.	VSCCSID

Overview for MQCHARV

Availability: AIX, HP-UX, Solaris, Linux, i5/OS, Windows, plus WebSphere MQ clients connected to these systems.

Purpose: Use the MQCHARV structure to describe a variable length string.

Character set and encoding: Data in the MQCHARV must be in the encoding of the local queue manager that is given by MQENC_NATIVE and the character set of the VSCCSID field within the structure. If the application is running as an MQ client, the structure must be in the encoding of the client. Some character sets have a representation that depends on the encoding. If VSCCSID is one of these character sets, the encoding used is the same encoding as that of the other fields in the MQCHARV.

Usage: The MQCHARV structure addresses data that might be discontinuous with the structure containing it. To address this data, fields declared with the pointer data type can be used. Be aware that COBOL does not support the pointer data type in all environments. Because of this, the data can also be addressed using fields that contain the offset of the data from the start of the structure containing the MQCHARV.

COBOL programming

If you want to port an application between environments, you must ascertain whether the pointer data type is available in all the intended environments. If not, the application must address the data using the offset fields instead of the pointer fields.

In those environments where pointers are not supported, you can declare the pointer fields as byte strings of the appropriate length, with the initial value being

the all-null byte string. Do not alter this initial value if you are using the offset fields. One way to do this without changing the supplied copy books is to use the following:

```
COPY CMQCHRVV REPLACING POINTER BY ==BINARY PIC S9(9)==.
```

where CMQCHRVV can be exchanged for the copy book to be used.

Fields for MQCHARV

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

VSBufSize (MQLONG)

This is the size in bytes of the buffer addressed by the VSPtr or VSOOffset field.

When the MQCHARV structure is used as an output field on a function call, this field must be initialised with the length of the buffer provided. If the value of VSLength is greater than VSBufSize then only VSBufSize bytes of data are returned to the caller in the buffer.

This value must be a value greater than or equal to zero, or the following special value which is recognized:

MQVS_USE_VSLENGTH

When specified, the length of the buffer is taken from the VSLength field in the MQCHARV structure. Do not use this value when using the structure as an output field and a buffer is provided.

This is the initial value of this field.

VSCCSID (MQLONG)

This is the character set identifier of the variable length string addressed by the VSPtr or VSOOffset field.

The initial value of this field is MQCCSI_APPL which is defined by MQ to indicate that it should be changed to the true character set identifier of the current process. As a result, the value MQCCSI_APPL is never associated with a variable length string. The initial value of this field can be changed by defining a different value for the constant MQCCSI_APPL for your compile unit by the appropriate means for your application's programming language.

VSLength (MQLONG)

The length in bytes of the variable length string addressed by the VSPtr or VSOOffset field.

The initial value of this field is 0. The value must be either greater than or equal to zero or the following special value which is recognized:

MQVS_NULL_TERMINATED

If MQVS_NULL_TERMINATED is not specified, VSLength bytes are included as part of the string. If null characters are present they do not delimit the string.

If MQVS_NULL_TERMINATED is specified, the string is delimited by the first null encountered in the string. The null itself is not included as part of that string.

Note: The null character used to terminate a string if MQVS_NULL_TERMINATED is specified is a null from the codeset specified by VSCCSID.

For example, in UTF-16 (UCS-2 CCSIDs 1200 and 13488), this is the two byte Unicode encoding where a null is represented by a 16 bit number of all zeros. In UTF-16 it is common to find single bytes set to all zero which are part of characters (seven bit ASCII characters for instance), but the strings will only be null terminated when two 'zero' bytes are found on an even byte boundary. It is possible to get two 'zero' bytes on an odd boundary when they are each part of valid characters, for example x'01' x'00 x'00' x'30' would be two valid Unicode characters and would not null terminate the string.

VSOFFSET (MQLONG)

The offset can be positive or negative. You can use either the VSPtr or VSOFFSET field to specify the variable length string, but not both. The offset in bytes of the variable length string from the start of the MQCHARV, or the structure containing it.

When the MQCHARV structure is embedded within another structure, this value is the offset in bytes of the variable length string from the start of the structure that contains this MQCHARV structure. When the MQCHARV structure is not embedded within another structure, for example, if it is specified as a parameter on a function call, the offset is relative to the start of the MQCHARV structure.

The initial value of this field is 0.

VSPTR (MQPTR)

This is a pointer to the variable length string.

You can use either the VSPtr or VSOFFSET field to specify the variable length string, but not both.

The initial value of this field is a null pointer or null bytes.

Initial values and language declarations for MQCHARV

Initial values of fields in MQCHARV

Field name	Name of constant	Value of constant
<i>VSPtr</i>	None	Null pointer or null bytes.
<i>VSOFFSET</i>	None	0
<i>VSBuFSize</i>	MQVS_USE_VSLENGTH	-1
<i>VSLength</i>	None	0
<i>VSCCSID</i>	MQCCSI_APPL	-3

Note: In the C programming language, the macro variable MQCHARV_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQCHARV MyVarStr = {MQCHARV_DEFAULT};
```

C declaration for MQCHARV

```
typedef struct tagMQCHARV MQCHARV;
struct tagMQCHARV {
    MQPTR    VSPtr;           /* Address of variable length string */
    MQLONG   VSOFFSET;       /* Offset of variable length string */
    MQLONG   VSBufSize;      /* Size of buffer */
    MQLONG   VSLength;       /* Length of variable length string */
    MQLONG   VSCCSID;        /* CCSID of variable length string */
};
```

COBOL declaration for MQCHARV

```
** MQCHARV structure
10 MQCHARV.
** Address of variable length string
15 MQCHARV-VSPTR    POINTER.
** Offset of variable length string
15 MQCHARV-VSOFFSET PIC S9(9) BINARY.
** Size of buffer
15 MQCHARV-VSBUFSIZE PIC S9(9) BINARY.
** Length of variable length string
15 MQCHARV-VSLENGTH PIC S9(9) BINARY.
** CCSID of variable length string
15 MQCHARV-VSCCSID  PIC S9(9) BINARY.
```

PL/I declaration

```
dc1
1 MQCHARV based,
3 VSPtr    pointer,      /* Address of variable length string */
3 VSOFFSET fixed bin(31), /* Offset of variable length string */
3 VSBufSize fixed bin(31), /* Size of buffer */
3 VSLength fixed bin(31), /* Length of variable length string */
3 VSCCSID  fixed bin(31); /* CCSID of variable length string */
```

System/390 assembler declaration

```
MQCHARV          DSECT
MQCHARV_VSPTR    DS  F    Address of variable length string
MQCHARV_VSOFFSET DS  F    Offset of variable length string
MQCHARV_VSBUFSIZE DS  F    Size of buffer
MQCHARV_VSLENGTH DS  F    Length of variable length string
MQCHARV_VSCCSID DS  F    CCSID of variable length string
*
MQCHARV_LENGTH   EQU  *-MQCHARV
                  ORG  MQCHARV
MQCHARV_AREA     DS   CL(MQCHARV_LENGTH)
```

Redefinition of MQCCSI_APPL

The following examples show how you can override the value of MQCCSI_APPL in various programming languages. You can change the value of MQCCSI_APPL, removing the need to set the VSCCSID for each variable length string separately.

In these examples the CCSID is set to 1208; change this to the value you require. This becomes the default value, which you can override by setting the VSCCSID in any specific instance of MQCHARV.

C usage

```
#define MQCCSI_APPL 1208
#include <cmqc.h>
```

COBOL usage

```
COPY CMQXYZV REPLACING -3 BY 1208.
```

PL/I usage

```
%MQCCSI_APPL = '1208';  
%include syslib(cmqp);
```

System/390 assembler usage

```
MQCCSI_APPL EQU 1208  
CMQA LIST=NO
```

MQCIH – CICS bridge header

The following table summarizes the fields in the structure.

Table 16. Fields in MQCIH

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

Table 16. Fields in MQCIH (continued)

Field	Description	Topic
<i>StartCode</i>	Transaction start code	StartCode
<i>CancelCode</i>	Abend transaction code	CancelCode
<i>NextTransactionId</i>	Next transaction to attach	NextTransactionId
<i>Reserved2</i>	Reserved	Reserved2
<i>Reserved3</i>	Reserved	Reserved3
Note: The remaining fields are not present if <i>Version</i> is less than MQCIH_VERSION_2.		
<i>CursorPosition</i>	Cursor position	CursorPosition
<i>ErrorOffset</i>	Offset of error in message	ErrorOffset
<i>InputItem</i>	Reserved	InputItem
<i>Reserved4</i>	Reserved	Reserved4

Overview for MQCIH

Availability: AIX, HP-UX, z/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

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

Format name: MQFMT_CICS.

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

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

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

- Applications that connect to the queue manager that owns the CICS bridge queue must provide an MQCIH structure that is in the character set and encoding of the queue manager. This is because data conversion of the MQCIH structure is not performed in this case.
- Applications that connect to other queue managers can provide an MQCIH structure that is in any of the supported character sets and encodings; the receiving message channel agent connected to the queue manager that owns the CICS bridge queue converts the MQCIH structure.
- The application message data following the MQCIH structure must be in the same character set and encoding as the MQCIH structure. You cannot use the *CodedCharSetId* and *Encoding* fields in the MQCIH structure to specify the character set and encoding of the application message data.

You must provide a data-conversion exit to convert the application message data if the data is not one of the built-in formats supported by the queue manager.

Usage: If the application requires values that are the same as the initial values shown in Table 18 on page 69, and the bridge is running with AUTH=LOCAL or

AUTH=IDENTIFY, you can omit the MQCIH structure from the message. In all other cases, the structure must be present.

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

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

Fields documented as response fields are set by the CICS bridge in the reply message that the bridge sends to the application. Error information is returned in the *ReturnCode*, *Function*, *CompCode*, *Reason*, and *AbendCode* fields, but not all of them are set in all cases. Table 17 shows which fields are set for different values of *ReturnCode*.

Table 17. Contents of error information fields in MQCIH structure for MQCIH

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

Fields for MQCIH

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

AbendCode (MQCHAR4)

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

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

ADSDescriptor (MQLONG)

This is an indicator specifying whether to send ADS descriptors on SEND and RECEIVE BMS requests.

The following values are defined:

MQCADSD_NONE

Do not send or receive ADS descriptors.

MQCADSD_SEND

Send ADS descriptors.

MQCADSD_RECV

Receive ADS descriptors.

MQCADSD_MSGFORMAT

Use message format for the ADS descriptors.

This sends or receives the ADS descriptors using the long form of the ADS descriptor. The long form has fields that are aligned on 4-byte boundaries.

Set the *ADSDescriptor* field as follows:

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

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

AttentionId (MQCHAR4)

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

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

Authenticator (MQCHAR8)

This is a password or passticket.

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

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

CancelCode (MQCHAR4)

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

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

CodedCharSetId (MQLONG)

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

The Character Set Id for supported structures which follow a MQCIH structure is the same as that of the MQCIH structure itself and taken from any preceding MQ header.

CompCode (MQLONG)

The value returned in this field depends on *ReturnCode*; see Table 17 on page 59.

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

ConversationalTask (MQLONG)

This is an indicator specifying whether to allow the task to issue requests for more information, or to abend the task.

The value must be one of the following:

MQCCT_YES

Task is conversational.

MQCCT_NO

Task is not conversational.

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

CursorPosition (MQLONG)

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

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

Encoding (MQLONG)

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

| The Encoding for supported structures which follow a MQCIH structure is the
| same as that of the MQCIH structure itself and taken from any preceding MQ
| header.

ErrorOffset (MQLONG)

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

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

Facility (MQBYTE8)

This is an 8-byte bridge facility token.

A bridge facility token allows multiple transactions in a pseudo-conversation to use the same bridge facility (virtual 3270 terminal). In the first, or only, message in a pseudo-conversation, set a value of MQCFAC_NONE; this tells CICS to allocate a new bridge facility for this message. A bridge facility token is returned in response messages when a nonzero *FacilityKeepTime* is specified on the input message. Subsequent input messages within a pseudo-conversation must then use the same bridge facility token.

The following special value is defined:

MQCFAC_NONE

No facility token specified.

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

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

FacilityKeepTime (MQLONG)

This is the length of time in seconds that the bridge facility is kept after the user transaction ends.

For pseudo-conversational transactions specify a value that corresponds to the expected duration of a pseudo-conversation; specify zero for the last transaction of a pseudo-conversation; for other transaction types specify zero.

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

FacilityLike (MQCHAR4)

This is the name of an installed terminal that is to be used as a model for the bridge facility.

A value of blanks means that *FacilityLike* is taken from the bridge transaction profile definition, or a default value is used.

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

Flags (MQLONG)

The value must be:

MQCIH_NONE

No flags.

MQCIH_PASS_EXPIRATION

The reply message contains:

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

If you omit this value, the expiry time is set to *unlimited*.

MQCIH_REPLY_WITHOUT_NULLS

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

MQCIH_SYNC_ON_RETURN

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

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

Format (MQCHAR8)

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

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

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

- For DPL requests, *Format* must be the format name of the COMMAREA.
- For 3270 requests, *Format* must be CSQCBDCl, and the bridge sets the format to CSQCBDc0 for Reply messages.

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

If the request message generates an error reply message, the error reply message has a format name of MQFMT_STRING.

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

Function (MQCHAR4)

The value returned in this field depends on *ReturnCode*; see Table 17 on page 59. The following values are possible when *Function* contains an MQ call name:

MQCFUNC_MQCONN
MQCONN call.

MQCFUNC_MQGET
MQGET call.

MQCFUNC_MQINQ
MQINQ call.

MQCFUNC_MQOPEN
MQOPEN call.

MQCFUNC_MQPUT
MQPUT call.

MQCFUNC_MQPUT1
MQPUT1 call.

MQCFUNC_NONE
No call.

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

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

GetWaitInterval (MQLONG)

This field applies only when *UOWControl* has the value MQCUOWC_FIRST. It allows the sending application to specify the approximate time in milliseconds that

the MQGET calls issued by the bridge should wait for second and subsequent request messages for the unit of work started by this message. This overrides the default wait interval used by the bridge. You can use the following special values:

MQCGWI_DEFAULT

Default wait interval.

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

MQWI_UNLIMITED

Unlimited wait interval.

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

InputItem (MQLONG)

This is a reserved field.

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

LinkType (MQLONG)

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

MQCLT_PROGRAM

DPL program.

MQCLT_TRANSACTION

3270 transaction.

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

NextTransactionId (MQCHAR4)

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

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

OutputDataLength (MQLONG)

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

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

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

You can use the following special value:

MQCODL_AS_INPUT

Output length is same as input length.

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

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

Reason (MQLONG)

The value returned in this field depends on *ReturnCode*; see Table 17 on page 59.

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

RemoteSysId (MQCHAR4)

This is the CICS system identifier of the CICS system processing the request. If this field is blank, the CICS system request is processed on the same CICS system as the bridge monitor. The SYSID used is returned in the Reply message.

For a 3270 pseudo-conversation, all subsequent messages in the conversation must specify the remote SYSID returned in the initial reply. If specified, the SYSID must:

- Be active
- Have access to the WebSphere MQ Request queue
- Be accessible by the CICS ISC links from the bridge monitor's CICS system

RemoteTransId (MQCHAR4)

This is an optional Request field. The length of this field is given by MQ_TRANSACTION_ID_LENGTH. If specified, the field is used as the RTRANSID value of CICS START.

ReplyToFormat (MQCHAR8)

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

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

Reserved1 (MQCHAR8)

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

Reserved2 (MQCHAR8)

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

Reserved3 (MQCHAR8)

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

Reserved4 (MQLONG)

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

ReturnCode (MQLONG)

This is the return code from the CICS bridge describing the outcome of the processing performed by the bridge. The *Function*, *CompCode*, *Reason*, and *AbendCode* fields might contain additional information (see Table 17 on page 59). The value is one of the following:

MQCRC_APPLICATION_ABEND

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

MQCRC_BRIDGE_ABEND

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

MQCRC_BRIDGE_ERROR

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

MQCRC_BRIDGE_TIMEOUT

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

MQCRC_CICS_EXEC_ERROR

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

MQCRC_MQ_API_ERROR

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

MQCRC_OK

(0, X'000') No error.

MQCRC_PROGRAM_NOT_AVAILABLE

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

MQCRC_SECURITY_ERROR

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

MQCRC_TRANSID_NOT_AVAILABLE

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

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

StartCode (MQCHAR4)

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

MQCSC_START

Start.

MQCSC_STARTDATA

Start data.

MQCSC_TERMINPUT

Terminal input.

MQCSC_NONE

None.

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

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

- MQCSC_START
- MQCSC_STARTDATA
- MQCSC_TERMINPUT

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

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

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

StrucId (MQCHAR4)

The value must be:

MQCIH_STRUC_ID

Identifier for CICS information header structure.

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

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

StrucLength (MQLONG)

The value must be one of the following:

MQCIH_LENGTH_1

Length of version-1 CICS information header structure.

MQCIH_LENGTH_2

Length of version-2 CICS information header structure.

The following constant specifies the length of the current version:

MQCIH_CURRENT_LENGTH

Length of current version of CICS information header structure.

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

TaskEndStatus (MQLONG)

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

MQCTES_NOSYNC

Not synchronized.

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

MQCTES_COMMIT

Commit unit of work.

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

MQCTES_BACKOUT

Back out unit of work.

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

MQCTES_ENDTASK

End task.

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

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

TransactionId (MQCHAR4)

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

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

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

UOWControl (MQLONG)

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

The value must be one of the following:

MQCUOWC_ONLY

Start unit of work, perform function, then commit the unit of work.

MQCUOWC_CONTINUE

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

MQCUOWC_FIRST

Start unit of work and perform function.

MQCUOWC_MIDDLE

Perform function within current unit of work

MQCUOWC_LAST

Perform function, then commit the unit of work.

MQCUOWC_COMMIT

Commit the unit of work (DPL only).

MQCUOWC_BACKOUT

Back out the unit of work (DPL only).

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

Version (MQLONG)

The value must be one of the following:

MQCIH_VERSION_1

Version-1 CICS information header structure.

MQCIH_VERSION_2

Version-2 CICS information header structure.

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

MQCIH_CURRENT_VERSION

Current version of CICS information header structure.

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

Initial values and language declarations for MQCIH

Table 18. Initial values of fields in MQCIH for MQCIH

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

Table 18. Initial values of fields in MQCIH for MQCIH (continued)

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

C declaration

```
typedef struct tagMQCIH MQCIH;
struct tagMQCIH {
    MQCHAR4  StrucId;           /* Structure identifier */
    MQLONG   Version;          /* Structure version number */
    MQLONG   StrucLength;      /* Length of MQCIH structure */
    MQLONG   Encoding;         /* Reserved */
    MQLONG   CodedCharSetId;   /* Reserved */
    MQCHAR8  Format;           /* MQ format name of data that follows
                               MQCIH */
    MQLONG   Flags;            /* Flags */
    MQLONG   ReturnCode;       /* Return code from bridge */
    MQLONG   CompCode;         /* MQ completion code or CICS EIBRESP */
    MQLONG   Reason;          /* MQ reason or feedback code, or CICS
                               EIBRESP2 */
    MQLONG   UOWControl;       /* Unit-of-work control */
};
```

```

MQLONG  GetWaitInterval;    /* Wait interval for MQGET call issued
                             by bridge task */
MQLONG  LinkType;          /* Link type */
MQLONG  OutputDataLength;  /* Output COMMAREA data length */
MQLONG  FacilityKeepTime;  /* Bridge facility release time */
MQLONG  ADSDescriptor;     /* Send/receive ADS descriptor */
MQLONG  ConversationalTask; /* Whether task can be conversational */
MQLONG  TaskEndStatus;     /* Status at end of task */
MQBYTE8 Facility;         /* Bridge facility token */
MQCHAR4 Function;         /* MQ call name or CICS EIBFN
                             function */

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

```

COBOL declaration

```

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

```

```

15 MQCIH-CONVERSATIONALTASK PIC S9(9) BINARY.
** Status at end of task
15 MQCIH-TASKENDSTATUS PIC S9(9) BINARY.
** Bridge facility token
15 MQCIH-FACILITY PIC X(8).
** MQ call name or CICS EIBFN function
15 MQCIH-FUNCTION PIC X(4).
** Abend code
15 MQCIH-ABENDCODE PIC X(4).
** Password or passticket
15 MQCIH-AUTHENTICATOR PIC X(8).
** Reserved
15 MQCIH-RESERVED1 PIC X(8).
** MQ format name of reply message
15 MQCIH-REPLYTOFORMAT PIC X(8).
** Reserved
15 MQCIH-REMOTESYSID PIC X(4).
** Reserved
15 MQCIH-REMOETTRANSID PIC X(4).
** Transaction to attach
15 MQCIH-TRANSACTIONID PIC X(4).
** Terminal emulated attributes
15 MQCIH-FACILITYLIKE PIC X(4).
** AID key
15 MQCIH-ATTENTIONID PIC X(4).
** Transaction start code
15 MQCIH-STARTCODE PIC X(4).
** Abend transaction code
15 MQCIH-CANCELCODE PIC X(4).
** Next transaction to attach
15 MQCIH-NEXTTRANSACTIONID PIC X(4).
** Reserved
15 MQCIH-RESERVED2 PIC X(8).
** Reserved
15 MQCIH-RESERVED3 PIC X(8).
** Cursor position
15 MQCIH-CURSORPOSITION PIC S9(9) BINARY.
** Offset of error in message
15 MQCIH-ERROROFFSET PIC S9(9) BINARY.
** Reserved
15 MQCIH-INPUTITEM PIC S9(9) BINARY.
** Reserved
15 MQCIH-RESERVED4 PIC S9(9) BINARY.

```

PL/I declaration

```

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

```

3	ConversationalTask	fixed bin(31),	/* Whether task can be conversational */
3	TaskEndStatus	fixed bin(31),	/* Status at end of task */
3	Facility	char(8),	/* Bridge facility token */
3	Function	char(4),	/* MQ call name or CICS EIBFN function */
3	AbendCode	char(4),	/* Abend code */
3	Authenticator	char(8),	/* Password or passticket */
3	Reserved1	char(8),	/* Reserved */
3	ReplyToFormat	char(8),	/* MQ format name of reply message */
3	RemoteSysId	char(4),	/* Reserved */
3	RemoteTransId	char(4),	/* Reserved */
3	TransactionId	char(4),	/* Transaction to attach */
3	FacilityLike	char(4),	/* Terminal emulated attributes */
3	AttentionId	char(4),	/* AID key */
3	StartCode	char(4),	/* Transaction start code */
3	CancelCode	char(4),	/* Abend transaction code */
3	NextTransactionId	char(4),	/* Next transaction to attach */
3	Reserved2	char(8),	/* Reserved */
3	Reserved3	char(8),	/* Reserved */
3	CursorPosition	fixed bin(31),	/* Cursor position */
3	ErrorOffset	fixed bin(31),	/* Offset of error in message */
3	InputItem	fixed bin(31),	/* Reserved */
3	Reserved4	fixed bin(31);	/* Reserved */

System/390 assembler declaration

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

```

MQCIH_INPUTITEM      DS   F   Reserved
MQCIH_RESERVED4     DS   F   Reserved
*
MQCIH_LENGTH         EQU  *-MQCIH
                     ORG  MQCIH
MQCIH_AREA           DS   CL(MQCIH_LENGTH)

```

Visual Basic declaration

```

Type MQCIH
  StrucId           As String*4 'Structure identifier'
  Version           As Long      'Structure version number'
  StrucLength       As Long      'Length of MQCIH structure'
  Encoding          As Long      'Reserved'
  CodedCharSetId    As Long      'Reserved'
  Format            As String*8  'MQ format name of data that follows'
                    'MQCIH'

  Flags            As Long      'Flags'
  ReturnCode        As Long      'Return code from bridge'
  CompCode          As Long      'MQ completion code or CICS EIBRESP'
  Reason            As Long      'MQ reason or feedback code, or CICS'
                    'EIBRESP2'

  UOWControl        As Long      'Unit-of-work control'
  GetWaitInterval   As Long      'Wait interval for MQGET call issued'
                    'by bridge task'

  LinkType          As Long      'Link type'
  OutputDataLength  As Long      'Output COMMAREA data length'
  FacilityKeepTime  As Long      'Bridge facility release time'
  ADSDescriptor     As Long      'Send/receive ADS descriptor'
  ConversationalTask As Long      'Whether task can be conversational'
  TaskEndStatus     As Long      'Status at end of task'
  Facility          As MQBYTE8   'Bridge facility token'
  Function          As String*4   'MQ call name or CICS EIBFN function'
  AbendCode         As String*4   'Abend code'
  Authenticator     As String*8   'Password or passticket'
  Reserved1         As String*8   'Reserved'
  ReplyToFormat     As String*8   'MQ format name of reply message'
  RemoteSysId       As String*4   'Reserved'
  RemoteTransId     As String*4   'Reserved'
  TransactionId     As String*4   'Transaction to attach'
  FacilityLike      As String*4   'Terminal emulated attributes'
  AttentionId       As String*4   'AID key'
  StartCode         As String*4   'Transaction start code'
  CancelCode        As String*4   'Abend transaction code'
  NextTransactionId As String*4   'Next transaction to attach'
  Reserved2         As String*8   'Reserved'
  Reserved3         As String*8   'Reserved'
  CursorPosition    As Long      'Cursor position'
  ErrorOffset       As Long      'Offset of error in message'
  InputItem         As Long      'Reserved'
  Reserved4         As Long      'Reserved'
End Type

```

MQCMHO – Create-message options

The following table summarizes the fields in the structure.

Table 19. Fields in MQCMHO

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>Options</i>	Options	Options

Overview for MQCMHO

Availability: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, z/OS and WebSphere MQ clients.

Purpose: The MQCMHO structure allows applications to specify options that control how message handles are created. The structure is an input parameter on the MQCRTMH call.

Character set and encoding: Data in MQCMHO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields for MQCMHO

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

Options (MQLONG)

One of the following options can be specified:

MQCMHO_VALIDATE

When MQSETMP is called to set a property in this message handle, the property name will be validated to ensure that it:

- contains no invalid characters.
- does not begin "JMS" or "usr.JMS" except for the following:
 - JMSCorrelationID
 - JMSReplyTo
 - JMSType
 - JMSXGroupID
 - JMSXGroupSeq

These names are reserved for JMS properties.

- is not one of the following keywords, in any mixture of upper or lowercase:
 - "AND"
 - "BETWEEN"
 - "ESCAPE"
 - "FALSE"
 - "IN"
 - "IS"
 - "LIKE"
 - "NOT"
 - "NULL"
 - "OR"
 - "TRUE"
- does not begin "Body." or "Root." (except for "Root.MQMD.").

If the property is MQ-defined ("mq.*") and the name is recognized, the property descriptor fields will be set to the correct values for the property. If the property is not recognized, the *Support* field of the property descriptor is set to MQPD_OPTIONAL.

MQCMHO_DEFAULT_VALIDATION

This specifies that the default level of validation of property names should occur.

The default level of validation is equivalent to that specified by MQCMHO_VALIDATE.

In a future release an administrative option may be defined which will change the level of validation that will occur when MQCMHO_DEFAULT_VALIDATION is defined.

This is the default value.

MQCMHO_NO_VALIDATION

No validation on the property name will occur. See the description of MQCMHO_VALIDATE.

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

MQCMHO_NONE

All options assume their default values. Use this value to indicate that no other options have been specified. MQCMHO_NONE aids 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 always an input field. The initial value of this field is MQCMHO_DEFAULT_VALIDATION.

StrucId (MQCHAR4)

This is the structure identifier; the value must be:

MQCMHO_STRUC_ID

Identifier for create message handle options structure.

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

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

Version (MQLONG)

This is the structure version number; the value must be:

MQCMHO_VERSION_1

Version-1 create message handle options structure.

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

MQCMHO_CURRENT_VERSION

Current version of create message handle options structure.

This is always an input field. The initial value of this field is MQCMHO_VERSION_1.

Initial values and language declarations for MQCMHO

Table 20. Initial values of fields in MQCMHO

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQCMHO_STRUC_ID	'CMHO'
<i>Version</i>	MQCMHO_VERSION_1	1
<i>Options</i>	MQCMHO_DEFAULT_VALIDATION	0
Notes:		
1. In the C programming language, the macro variable MQCMHO_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure: <pre>MQCMHO MyCMHO = {MQCMHO_DEFAULT};</pre>		

C declaration

```
struct tagMQCMHO {
    MQCHAR4  StrucId;      /* Structure identifier */
    MQLONG   Version;     /* Structure version number */
    MQLONG   Options;     /* Options that control the action of MQCRTMH */
};
```

COBOL declaration

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

PL/I declaration

```
dcl
  1 MQCMHO based,
  3 StrucId      char(4),      /* Structure identifier */
  3 Version      fixed bin(31), /* Structure version number */
  3 Options      fixed bin(31), /* Options that control the action of MQCRTMH */
```

System/390 assembler declaration

```
MQCMHO          DSECT
MQCMHO_STRUCID  DS CL4  Structure identifier
MQCMHO_VERSION  DS F    Structure version number
MQCMHO_OPTIONS  DS F    Options that control the action of
*                MQCRTMH
MQCMHO_LENGTH   EQU *-MQCMHO
MQCMHO_AREA     DS CL(MQCMHO_LENGTH)
```

MQCNO – Connect options

The following table summarizes the fields in the structure.

Table 21. Fields in MQCNO

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version

Table 21. Fields in MQCNO (continued)

Field	Description	Topic
<i>Options</i>	Options that control the action of MQCONNX	Options
Note: The remaining fields are ignored if <i>Version</i> is less than MQCNO_VERSION_2.		
<i>ClientConnOffset</i>	Offset of MQCD structure for client connection	ClientConnOffset
<i>ClientConnPtr</i>	Address of MQCD structure for client connection	ClientConnPtr
Note: The remaining fields are ignored if <i>Version</i> is less than MQCNO_VERSION_3.		
<i>ConnTag</i>	Queue-manager connection tag	ConnTag
Note: The remaining fields are ignored if <i>Version</i> is less than MQCNO_VERSION_4.		
<i>SSLConfigPtr</i>	Address of MQSCO structure for client connection	SSLConfigPtr
<i>SSLConfigOffset</i>	Offset of MQSCO structure for client connection	SSLConfigOffset
Note: The remaining fields are ignored if <i>Version</i> is less than MQCNO_VERSION_5.		
<i>ConnectionId</i>	Unique connection ID	ConnectionId
<i>SecurityParmsOffset</i>	Security parameters	SecurityParmsOffset
<i>SecurityParmsPtr</i>	Security parameters	SecurityParmsPtr

Overview for MQCNO

Availability: All versions except MQCNO_VERSION_4: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

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

See the *WebSphere MQ Application Programming Guide* for details of using shared handles within a global unit of work and the effect that this has on, for example, XA transactions.

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

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

Fields for MQCNO

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

ClientConnOffset (MQLONG)

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

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

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

ClientConnPtr (MQPTR)

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

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

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

If the application provides an MQCD structure, set the fields listed below to the values required; other fields in MQCD are ignored. You can pad character strings with blanks to the length of the field, or terminated them with a null character. Refer to *WebSphere MQ Intercommunications* for more information about the fields in the MQCD structure.

Field in MQCD	Value
<i>ChannelName</i>	Channel name.
<i>Version</i>	Structure version number. Must not be less than MQCD_VERSION_7.
<i>TransportType</i>	Any supported transport type.
<i>ModeName</i>	LU 6.2 mode name.
<i>TpName</i>	LU 6.2 transaction program name.
<i>SecurityExit</i>	Name of channel security exit.
<i>SendExit</i>	Name of channel send exit.
<i>ReceiveExit</i>	Name of channel receive exit.
<i>MaxMsgLength</i>	Maximum length in bytes of messages that can be sent over the client connection channel.
<i>SecurityUserData</i>	User data for security exit.
<i>SendUserData</i>	User data for send exit.
<i>ReceiveUserData</i>	User data for receive exit.
<i>UserIdentifier</i>	User identifier to be used to establish an LU 6.2 session.
<i>Password</i>	Password to be used to establish an LU 6.2 session.
<i>ConnectionName</i>	Connection name.
<i>HeartbeatInterval</i>	Time in seconds between heartbeat flows.
<i>StrucLength</i>	Length of the MQCD structure.
<i>ExitNameLength</i>	Length of exit names addressed by <i>SendExitPtr</i> and <i>ReceiveExitPtr</i> . Must be greater than zero if <i>SendExitPtr</i> or <i>ReceiveExitPtr</i> is set to a value that is not the null pointer.

Field in MQCD	Value
<i>ExitDataLength</i>	Length of exit data addressed by <i>SendUserDataPtr</i> and <i>ReceiveUserDataPtr</i> . Must be greater than zero if <i>SendUserDataPtr</i> or <i>ReceiveUserDataPtr</i> is set to a value that is not the null pointer.
<i>SendExitsDefined</i>	Number of send exits addressed by <i>SendExitPtr</i> . If zero, <i>SendExit</i> and <i>SendUserData</i> provide the exit name and data. If greater than zero, <i>SendExitPtr</i> and <i>SendUserDataPtr</i> provide the exit names and data, and <i>SendExit</i> and <i>SendUserData</i> must be blank.
<i>ReceiveExitsDefined</i>	Number of receive exits addressed by <i>ReceiveExitPtr</i> . If zero, <i>ReceiveExit</i> and <i>ReceiveUserData</i> provide the exit name and data. If greater than zero, <i>ReceiveExitPtr</i> and <i>ReceiveUserDataPtr</i> provide the exit names and data, and <i>ReceiveExit</i> and <i>ReceiveUserData</i> must be blank.
<i>SendExitPtr</i>	Address of name of first send exit.
<i>SendUserDataPtr</i>	Address of data for first send exit.
<i>ReceiveExitPtr</i>	Address of name of first receive exit.
<i>ReceiveUserDataPtr</i>	Address of data for first receive exit.
<i>LongRemoteUserIdLength</i>	Length of long remote user identifier.
<i>LongRemoteUserIdPtr</i>	Address of long remote user identifier.
<i>RemoteSecurityId</i>	Remote security identifier.
<i>SSLCipherSpec</i>	SSL CipherSpec.
<i>SSLPeerNamePtr</i>	Address of SSL peer name.
<i>SSLPeerNameLength</i>	Length of SSL peer name.
<i>KeepAliveInterval</i>	Value passed to the communications stack for keepalive timing for the channel
<i>LocalAddress</i>	The local communications address, including the IP address of the local network adapter to use, and a range of ports to use for outgoing connections.

Provide the channel definition structure in one of two ways:

- By using the offset field *ClientConnOffset*
 In this case, the application must declare a compound structure containing an MQCNO followed by the channel definition structure MQCD, and set *ClientConnOffset* to the offset of the channel definition structure from the start of the MQCNO. Ensure that this offset is correct. *ClientConnPtr* must be set to the null pointer or null bytes.
 Use *ClientConnOffset* for programming languages that do not support the pointer data type, or that implement the pointer data type in a way that is not portable to different environments (for example, the COBOL programming language).
 For the Visual Basic programming language, a compound structure called MQCNOCD is provided in the header file CMQXB.BAS; this structure contains an MQCNO structure followed by an MQCD structure. Initialize MQCNOCD by invoking the MQCNOCD_DEFAULTS subroutine. MQCNOCD is used with the MQCONNXAny variant of the MQCONNX call; see the description of the MQCONNX call for further details.
- By using the pointer field *ClientConnPtr*
 In this case, the application can declare the channel definition structure separately from the MQCNO structure, and set *ClientConnPtr* to the address of the channel definition structure. Set *ClientConnOffset* to zero.

Use *ClientConnPtr* for programming languages that support the pointer data type in a way that is portable to different environments (for example, the C programming language).

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

Whichever technique you choose, you can use only one of *ClientConnOffset* and *ClientConnPtr*; the call fails with reason code `MQRC_CLIENT_CONN_ERROR` if both are nonzero.

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

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

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

ConnectionId (MQBYTE24)

This output parameter is a unique 24-byte identifier that allows MQ to reliably identify an application. An application can use this identifier for correlation in `PUT` and `GET` calls.

The queue manager assigns a unique ID to all connections, however they are established. If an `MQCONN` establishes the connection with a version 5 `MQCNO`, the application can determine the `ConnectionId` from the returned `MQCNO`. The assigned identifier is guaranteed to be unique among all other identifiers that MQ generates, such as `CorrelId`, `MsgID`, and `GroupId`.

Use the `ConnectionId` to identify long running units of work using the PCF command `Inquire Connection` or the MQSC command `DISPLAY CONN`. The `ConnectionId` used by MQSC commands (`CONN`) is derived from the `ConnectionId` returned here. The PCF `Inquire` and `Stop Connection` commands can use the `ConnectionId` returned here without modification.

You can use the `ConnectionId` to force the end of a long running unit of work, by specifying the `ConnectionId` using the PCF command `Stop Connection` or the MQSC command `STOP CONN`. See *WebSphere MQ Programmable Command Formats and Administration Interface* and *WebSphere MQ Script (MQSC) Command Reference* for more information on using these commands.

The initial value of this field is 24 null bytes in all programming languages.

This field is not returned if *Version* is less than `MQCNO_VERSION_5`.

The length of this field is given by `MQ_CONNECTION_ID_LENGTH`.

ConnTag (MQBYTE128)

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

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

Use the following special value if you require no tag:

MQCT_NONE

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

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

This field is used when connecting to a z/OS queue manager. In other environments, specify the value MQCT_NONE.

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

Options (MQLONG)

Options that control the action of MQCONN.

Accounting options: The following options further control the type of accounting if the *AccountingConnOverride* queue manager attribute is set to MQMON_ENABLED:

MQCNO_ACCOUNTING_MQI_ENABLED

When monitoring data collection is switched off in the queue manager definition by setting the *MQIAccounting* attribute to MQMON_OFF, setting this flag enables MQI accounting data collection.

MQCNO_ACCOUNTING_MQI_DISABLED

When monitoring data collection is switched off in the queue manager definition by setting the *MQIAccounting* attribute to MQMON_OFF, setting this flag stops MQI accounting data collection.

MQCNO_ACCOUNTING_Q_ENABLED

When queue-accounting data collection is switched off in the queue manager definition by setting the *MQIAccounting* attribute to MQMON_OFF, setting this flag enables accounting data collection for those queues that specify queue manager in the *MQIAccounting* field of their queue definition.

MQCNO_ACCOUNTING_Q_DISABLED

When queue-accounting data collection is switched off in the queue manager definition by setting the *MQIAccounting* attribute to MQMON_OFF, setting this flag switches off accounting data collection for those queues that specify queue manager in the *MQIAccounting* field of their queue definition.

If none of these flags is defined, the accounting for the connection is as defined in the Queue Manager attributes.

Binding options: The following options control the type of MQ binding to use; specify only one of these options:

MQCNO_STANDARD_BINDING

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

If the queue manager supports multiple binding types, and you set MQCNO_STANDARD_BINDING, the queue manager looks for the *DefaultBindType* attribute in the *Connection* stanza in the *qm.ini* file (or the equivalent Windows registry entry) to select the actual type of binding. If this stanza is not defined or the value cannot be used or is not appropriate for the application, the queue manager selects an appropriate binding type. The queue manager sets the actual binding type used in the connect options.

Use MQCNO_STANDARD_BINDING in situations where the application might not have been fully tested, or might be unreliable or untrustworthy. MQCNO_STANDARD_BINDING is the default.

This option is supported in all environments.

MQCNO_FASTPATH_BINDING

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

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

MQCNO_FASTPATH_BINDING may be of advantage in situations where the use of multiple processes is a significant performance overhead compared to the overall resource used by the application. An application that uses the fastpath binding is known as a *trusted application*.

Consider the following important points when deciding whether to use the fastpath binding:

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

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

- On i5/OS, the job must run under a user profile that belongs to the QMQADM group. Also, the program must not terminate abnormally, otherwise unpredictable results may occur.
- On UNIX systems, the mqm user identifier and the mqm group identifier must be the effective user identifier and group identifier respectively. To make the application run in this way, configure the program so that it is owned by the mqm user identifier and mqm group identifier, and then set the setuid and setgid permission bits on the program.

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

- On Windows, the program must be a member of the mqm group.
- On Windows, fastpath binding is not supported for 64-bit applications.

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

This option is supported in the following environments: AIX, HP-UX,, i5/OS, Solaris, Linux, Windows. On z/OS the option is accepted but ignored.

MQCNO_SHARED_BINDING

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

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

MQCNO_ISOLATED_BINDING

The application and the local queue manager agent (the component that manages queuing operations) run in separate units of execution (generally, in separate processes). This arrangement maintains the integrity of the queue manager, that is, it protects the queue manager from errant programs. The application process and the local queue manager agent are isolated from each other in that they do not share resources.

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

On AIX, HP-UX, Solaris, Linux, and Windows, you can use the environment variable MQ_CONNECT_TYPE with the bind type specified by the *Options* field, to control the type of binding used. If you specify this environment variable, it must have the value FASTPATH, STANDARD; if it has some other value, it is ignored. The value of the environment variable is case sensitive.

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

- If you omit the environment variable, or give it a value that is not supported, use of the fastpath binding is determined solely by the *Options* field.

- If you give the environment variable a supported value, the fastpath binding is used only if *both* the environment variable and *Options* field specify the fastpath binding.

Connection-tag options: The following options control the use of the connection tag *ConnTag*. You can specify only one of these options.

- These options are supported only when connecting to a z/OS queue manager.

MQCNO_SERIALIZE_CONN_TAG_Q_MGR

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

MQCNO_SERIALIZE_CONN_TAG_QSG

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

MQCNO_RESTRICT_CONN_TAG_Q_MGR

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

- On z/OS, applications must run within the same MVS™ address space in order to share the connection tag. If the application using the connection tag is a client application, MQCNO_RESTRICT_CONN_TAG_Q_MGR is not allowed.

MQCNO_RESTRICT_CONN_TAG_QSG

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

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

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

- On z/OS, applications must run within the same MVS address space in order to share the connection tag. If the application using the connection tag is a client application, MQCNO_RESTRICT_CONN_TAG_QSG is not allowed.

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

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

- These options are supported in the following environments: AIX, HP-UX, i5/OS, Solaris, Linux, Windows.

MQCNO_HANDLE_SHARE_NONE

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

MQCNO_HANDLE_SHARE_BLOCK

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

MQCNO_HANDLE_SHARE_NO_BLOCK

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

A thread can have zero or one nonshared handle:

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

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

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

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

Sharing conversation options: The following options apply only to TCP/IP client connections. For SNA, SPX and NetBios channels, these values are ignored and the channel runs as in previous versions of the product:

MQCNO_NO_CONV_SHARING

This option does not permit conversation sharing and the connection must be the only conversation on that channel instance.

You might use MQCNO_NO_CONV_SHARING in situations where conversations are very heavily loaded and, therefore, where contention is a possibility on the server-connection end of the channel instance on which the sharing conversations exist.

MQCNO_ALL_CONVS_SHARE

This option permits conversation sharing; the application does not place any limit on the number of connections on the channel instance. This option is the default value.

If the application indicates that the channel instance can share, but the *SharingConversations* (SHARECNV) definition on the server-connection end of the channel is set to one, no sharing occurs and no warning is given to the application.

Similarly, if the application indicates that sharing is permitted but the server-connection *SharingConversations* definition is set to zero, no warning is given, and the application exhibits the same behavior as a client in versions of the product earlier than version 7.0; the application setting relating to sharing conversations is ignored.

MQCNO_NO_CONV_SHARING and MQCNO_ALL_CONVS_SHARE are mutually exclusive. If both options are specified on a particular connection, the connection is rejected with a reason code of MQRC_OPTIONS_ERROR.

Channel definition options: The following options control the use of the channel definition structure passed in the MQCNO:

MQCNO_CD_FOR_OUTPUT_ONLY

The channel definition structure in the MQCNO should only be used for output to return the channel name used on a successful MQCONN call.

If a valid channel definition structure is not provided then the call will fail with the reason code MQRC_CD_ERROR.

If the application is not running as a client the option is ignored.

The returned channel name can be used on a subsequent MQCONN call using the MQCNO_USE_CD_SELECTION option to reconnect using the same channel definition. This can be useful when there are multiple applicable channel definitions in the client channel table.

MQCNO_USE_CD_SELECTION

The MQCONN call should connect using the channel name contained in the channel definition structure passed in the MQCNO.

If the MQSERVER environment variable is set, the channel definition defined by it is used. If MQSERVER is not set, the client channel table is used.

If a channel definition with matching channel name and queue manager name is not found then the call will fail with reason code MQRC_Q_MGR_NAME_ERROR.

If a valid channel definition structure is not provided then the call will fail with the reason code MQRC_CD_ERROR.

If the application is not running as a client the option is ignored.

Default option: If you require none of the options described above, you can use the following option:

MQCNO_NONE

No options specified.

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

SecurityParmsOffset (MQLONG)

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

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQCNO_VERSION_5.

The MQCSP structure is defined in “MQCSP – Security parameters” on page 92.

SecurityParmsPtr (PMQCSP)

The address of the MQSCP structure, used to specify a user ID and password for authentication by the authorization service.

This is an input field. The initial value of this field is a null pointer or null bytes. This field is ignored if *Version* is less than MQCNO_VERSION_5.

The MQCSP structure is defined in “MQCSP – Security parameters” on page 92.

SSLConfigOffset (MQLONG)

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

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

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

SSLConfigPtr (PMQSCO)

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

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

The SSL information can be specified only on:

- The first MQCONN call of the client process, or
- A subsequent MQCONN call when all previous SSL/TLS connections to the queue manager have been concluded using MQDISC.

These are the only states in which the process-wide SSL environment can be initialized. If an MQCONNX call is issued specifying SSL information when the SSL environment already exists, the SSL information on the call is ignored and the connection is made using the existing SSL environment; the call returns completion code MQCC_WARNING and reason code MQRC_SSL_ALREADY_INITIALIZED in this case.

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

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

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

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

Strucl (MQCHAR4)

The value must be:

MQCNO_STRUC_ID

Identifier for connect-options structure.

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

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

Version (MQLONG)

The value must be one of the following:

MQCNO_VERSION_1

Version-1 connect-options structure.

MQCNO_VERSION_2

Version-2 connect-options structure.

MQCNO_VERSION_3

Version-3 connect-options structure.

MQCNO_VERSION_4

Version-4 connect-options structure.

MQCNO_VERSION_5

Version-5 connect-options structure.

This version of the MQCNO structure extends MQCNO_VERSION_3 on z/OS, and MQCNO_VERSION_4 on all other platforms.

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

MQCNO_CURRENT_VERSION

Current version of connect-options structure.

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

Initial values and language declarations for MQCNO

Table 22. Initial values of fields in MQCNO for MQCNO

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQCNO_STRUC_ID	'CNOB'
<i>Version</i>	MQCNO_VERSION_1	1
<i>Options</i>	MQCNO_NONE	0
<i>ClientConnOffset</i>	None	0
<i>ClientConnPtr</i>	None	Null pointer or null bytes
<i>ConnTag</i>	MQCT_NONE	Nulls
<i>SSLConfigPtr</i>	None	Null pointer or null bytes
<i>SSLConfigOffset</i>	None	0
<i>ConnectionId</i>	None	Null pointer or null bytes
<i>SecurityParmsOffset</i>	None	Null pointer or null bytes
<i>SecurityParmsPtr</i>	None	Null pointer or null bytes
Notes:		
1. The symbol <i>b</i> represents a single blank character.		
2. In the C programming language, the macro variable MQCNO_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:		
<pre>MQCNO MyCNO = {MQCNO_DEFAULT};</pre>		

C declaration

```
typedef struct tagMQCNO MQCNO;
struct tagMQCNO {
    MQCHAR4    StrucId;          /* Structure identifier */
    MQLONG     Version;         /* Structure version number */
    MQLONG     Options;         /* Options that control the action of
                                MQCONN */
    MQLONG     ClientConnOffset; /* Offset of MQCD structure for client
                                connection */
    MQPTR      ClientConnPtr;   /* Address of MQCD structure for client
                                connection */
    MQBYTE128  ConnTag;         /* Queue-manager connection tag */
    PMQSCO     SSLConfigPtr;    /* Address of MQSCO structure for client
                                connection */
    MQLONG     SSLConfigOffset; /* Offset of MQSCO structure for client
                                connection */
    MQBYTE24   ConnectionId;    /* Unique connection identifier */
    MQLONG     SecurityParmsOffset /* Security fields */
    PMQCSP     SecurityParmsPtr /* Security parameters */
};
```

COBOL declaration

```
** MQCNO structure
10 MQCNO.
**   Structure identifier
15 MQCNO-STRUCID      PIC X(4).
**   Structure version number
15 MQCNO-VERSION     PIC S9(9) BINARY.
**   Options that control the action of MQCONN
15 MQCNO-OPTIONS     PIC S9(9) BINARY.
**   Offset of MQCD structure for client connection
15 MQCNO-CLIENTCONNOFFSET PIC S9(9) BINARY.
**   Address of MQCD structure for client connection
15 MQCNO-CLIENTCONNPTR  POINTER.
**   Queue-manager connection tag
15 MQCNO-CONNTAG     PIC X(128).
**   Address of MQSCO structure for client connection
15 MQCNO-SSLCONFIGPTR  POINTER.
**   Offset of MQSCO structure for client connection
15 MQCNO-SSLCONFIGOFFSET PIC S9(9) BINARY.
**   Unique connection identifier
15 MQCNO-CONNECTIONID  PIC X(24).
**   Offset of MQCSP structure for security parameters
15 MQCNO-SECURITYPARMSOFFSET PIC S9(9) BINARY.
**   Address of MQCSP structure for security parameters
15 MQCNO-SECURITYPARMSPTR POINTER.
```

PL/I declaration

```
dcl
1 MQCNO based,
3 StrucId      char(4),      /* Structure identifier */
3 Version      fixed bin(31), /* Structure version number */
3 Options      fixed bin(31), /* Options that control the action
                             of MQCONN */
3 ClientConnOffset fixed bin(31), /* Offset of MQCD structure for
                             client connection */
3 ClientConnPtr  pointer,    /* Address of MQCD structure for
                             client connection */
3 ConnTag       char(128),   /* Queue-manager connection tag */
3 SSLConfigPtr  pointer,    /* Address of MQSCO structure for
                             client connection */
3 SSLConfigOffset fixed bin(31), /* Offset of MQSCO structure for
                             client connection */
3 ConnectionId  char(24),    /* Unique connection identifier
3 SecurityParmsOffset fixed bin(31); /* Offset of MQCSP structure for
                             security parameters */
3 SecurityParmsPtr pointer,  /* Address of MQCSP structure for
                             security parameters */
```

System/390 assembler declaration

```
MQCNO          DSECT
MQCNO_STRUCID  DS   CL4   Structure identifier
MQCNO_VERSION  DS   F     Structure version number
MQCNO_OPTIONS  DS   F     Options that control the action of
*                               MQCONN
MQCNO_CLIENTCONNOFFSET DS F     Offset of MQCD structure for client
*                               connection
MQCNO_CLIENTCONNPTR  DS   F     Address of MQCD structure for client
*                               connection
MQCNO_CONNTAG   DS   XL128 Queue-manager connection tag
*
MQCNO_CONNECTIONID DS   XL24 Unique connection identifier
*
MQCNO_SSLCONFIGOFFSET DS F     Offset of MQCSP structure for security
*                               parameters
MQCNO_SSLCONFIGPTR  DS   F     Address of MQCSP structure for security
*                               parameters
```



```

MQCNO_LENGTH      EQU  *-MQCNO
                  ORG  MQCNO
MQCNO_AREA        DS   CL(MQCNO_LENGTH)

```

Visual Basic declaration

```

Type MQCNO
  StrucId      As String*4  'Structure identifier'
  Version      As Long      'Structure version number'
  Options      As Long      'Options that control the action of'
                  'MQCONN'
  ClientConnOffset As Long  'Offset of MQCD structure for client'
                  'connection'
  ClientConnPtr  As MQPTR   'Address of MQCD structure for client'
                  'connection'
  ConnTag       As MBYTE128 'Queue-manager connection tag'
  SSLConfigPtr  As MQPTR   'Address of MQSCO structure for client'
                  'connection'
  SSLConfigOffset As Long  'Offset of MQSCO structure for client'
                  'connection'
  ConnectionId  As MBYTE24  'Unique connection identifier'
  SecurityParmsOffset As Long 'Offset of MQCSP structure for security'
                  'parameters'
  SecurityParmsPtr As MQPTR 'Address of MQCSP structure for security'
                  'parameters'
End Type

```

MQCSP – Security parameters

The following table summarizes the fields in the structure.

Table 23. Fields in MQCSP

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>AuthenticationType</i>	Type of authentication	AuthenticationType
<i>Reserved1</i>	Required for pointer alignment on i5/OS	Reserved1
<i>CSPUserIdPtr</i>	Address of user ID	CSPUserIdPtr
<i>CSPUserIdOffset</i>	Offset of user ID	CSPUserIdOffset
<i>CSPUserIdLength</i>	Length of user ID	CSPUserIdLength
<i>Reserved2</i>	Required for pointer alignment on i5/OS	Reserved2
<i>CSPPasswordPtr</i>	Address of password	CSPPasswordPtr
<i>CSPPasswordOffset</i>	Offset of password	CSPPasswordOffset
<i>CSPPasswordLength</i>	Length of password	CSPPasswordLength

Overview for MQCSP

Availability: All WebSphere MQ products.

Purpose: The MQCSP structure enables the authorization service to authenticate a user ID and password. You specify the MQCSP connection security parameters structure on an MQCONN call.

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

Fields for MQCSP

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

AuthenticationType (MQLONG)

This is the type of authentication to perform. Valid values are:

MQCSP_AUTH_NONE

Do not use user ID and password fields.

MQCSP_AUTH_USER_ID_AND_PWD

Authenticate user ID and password fields.

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

CSPPasswordLength (MQLONG)

This is the length of the password to be used in authentication.

The maximum length of the password is not dependent on the platform. If the length of the password is greater than that allowed, the authentication request fails with an MQRC_NOT_AUTHORIZED.

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

CSPPasswordOffset (MQLONG)

This is the offset in bytes of the password to be used in authentication. The offset can be positive or negative.

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

CSPPasswordPtr (MQPTR)

This is the address in bytes of the password to be used in authentication.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQCNO_VERSION_5.

CSPUserIdLength (MQLONG)

This is the length of the user ID to be used in authentication.

The maximum length of the user ID is not dependent on the platform. If the length of the user ID is greater than that allowed, the authentication request fails with an MQRC_NOT_AUTHORIZED.

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

CSPUserIdOffset (MQLONG)

This is the offset in bytes of the user ID to be used in authentication. The offset can be positive or negative.

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

CSPUserIntPtr (MQPTR)

This is the address in bytes of the user ID to be used in authentication.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQCNO_VERSION_5.

Reserved1 (MQBYTE4)

A reserved field, required for pointer alignment on i5/OS.

This is an input field. The initial value of this field is all null.

Reserved2 (MQBYTE8)

A reserved field, required for pointer alignment on i5/OS.

This is an input field. The initial value of this field is all null.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQCSP_STRUC_ID

Identifier for the security parameters structure.

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

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

Version (MQLONG)

Structure version number.

The value must be:

MQCSP_VERSION_1

Version-1 security parameters structure.

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

MQCSP_CURRENT_VERSION

Current version of security parameters structure.

This is always an input field. The initial value of this field is MQCSP_VERSION_1.

Initial values and language declarations for MQCSP

Table 24. Initial values of fields in MQCSP for MQCSP

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQCSP_STRUC_ID	'CSP'
<i>Version</i>	MQCSP_CURRENT_VERSION	1
<i>AuthenticationType</i>	None	MQCSP_AUTH_NONE
<i>Reserved1</i>	None	Null string or blanks

Table 24. Initial values of fields in MQCSP for MQCSP (continued)

Field name	Name of constant	Value of constant
<i>CSPUserIdPtr</i>	None	Null pointer or null bytes
<i>CSPUserIdOffset</i>	None	0
<i>CSPUserIdLength</i>	None	0
<i>Reserved2</i>	None	Null string or blanks
<i>CSPPasswordPtr</i>	None	Null pointer or null bytes
<i>CSPPasswordOffset</i>	None	0
<i>CSPPasswordLength</i>	None	0

Notes:

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

```
MQCSP MyCSP = {MQCSP_DEFAULT};
```

C declaration

```
typedef struct tagMQCSP MQCSP;
struct tagMQCSP {
    MQCHAR4    StrucId;           /* Structure identifier */
    MQLONG     Version;          /* Structure version number */
    MQLONG     AuthenticationType; /* Type of authentication */
    MQBYTE4    Reserved1;       /* Required for i5/OS pointer
                                alignment */
    MQPTR      CSPUserIdPtr;     /* Address of user ID */
    MQLONG     CSPUserIdOffset;  /* Offset of user ID */
    MQLONG     CSPUserIdLength;  /* Length of user ID */
    MQBYTE8    Reserved2;       /* Required for i5/OS pointer
                                alignment */
    MQPTR      CSPPasswordPtr;   /* Address of password */
    MQLONG     CSPPasswordOffset; /* Offset of password */
    MQLONG     CSPPasswordLength; /* Length of password */
};
```

COBOL declaration

```
** MQCSP structure
  10 MQCSP.
**   Structure identifier
  15 MQCSP-STRUCID          PIC X(4).
**   Structure version number
  15 MQCSP-VERSION        PIC S9(9) BINARY.
**   Type of authentication
  15 MQCSP-AUTHENTICATIONTYPE PIC S9(9) BINARY.
**   Required for i5/OS pointer alignment
  15 MQCSP-RESERVED1      PIC X(4).
**   Address of user ID
  15 MQCSP-CSPUSERIDPTR   POINTER.
**   Offset of user ID
  15 MQCSP-CSPUSERIDOFFSET PIC S9(9) BINARY.
**   Length of user ID
  15 MQCSP-CSPUSERIDLENGTH PIC S9(9) BINARY.
**   Required for i5/OS pointer alignment
  15 MQCSP-RESERVED2      PIC X(4).
```

```

**   Address of password
    15 MQCSP-CSPPASSWORDPTR   POINTER.
**   Offset of password
    15 MQCSP-CSPPASSWORDOFFSET PIC S9(9) BINARY.
**   Length of password
    15 MQCSP-CSPPASSWORDLENGTH PIC S9(9) BINARY.

```

PL/I declaration

```

dc1
1 MQCSP based,
3 StrucId          char(4),          /* Structure identifier */
3 Version          fixed bin(31), /* Structure version number */
3 AuthenticationType fixed bin(31), /* Type of authentication */
3 Reserved1       char(4),          /* Required for i5/OS pointer
                                     alignment */
3 CSPUserIdPtr    pointer,          /* Address of user ID */
3 CSPUserIdOffset fixed bin(31), /* Offset of user ID */
3 CSPUserIdLength fixed bin(31), /* Length of user ID */
3 Reserved2       char(8),          /* Required for i5/OS pointer
                                     alignment */
3 CSPPasswordPtr  pointer,          /* Address of password */
3 CSPPasswordOffset fixed bin(31), /* Offset of user ID */
3 CSPPasswordLength fixed bin(31); /* Length of user ID */

```

Visual Basic declaration

```

Type MQCSP
  StrucId          As String*4 'Structure identifier'
  Version          As Long     'Structure version number'
  AuthenticationType As Long   'Type of authentication'
  Reserved1       As MQBYTE4   'Required for i5/OS pointer'
                                     'alignment'
  CSPUserIdPtr    As MQPTR     'Address of user ID'
  CSPUserIdOffset As Long     'Offset of user ID'
  CSPUserIdLength As Long     'Length of user ID'
  Reserved2       As MQBYTE8   'Required for i5/OS pointer'
                                     'alignment'
  CSPPasswordPtr  As MQPTR     'Address of password'
  CSPPasswordOffset As Long   'Offset of password'
  CSPPasswordLength As Long   'Length of password'
End Type

```

MQCTLO – Control callback options structure

The following table summarizes the fields in the structure. Structure specifying the control callback function.

Table 25. Fields in MQCTLO

Field	Description	Topic
<i>StrucID</i>	Structure identifier	StrucID
<i>Version</i>	Structure version number	Version
<i>Options</i>	Options	Options
<i>Reserved</i>	Reserved field	Options
<i>ConnectionArea</i>	Field for callback function to use	ConnectionArea

Overview for MQCTLO

Availability: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, z/OS, and WebSphere MQ clients connected to these systems. Overview of the MQCTLO structure.

Purpose: The MQCTLO structure is used to specify options relating to a control callbacks function.

The structure is an input and output parameter on the MQCTL call.

Version: The current version of MQCTLO is MQCTLO_VERSION_1.

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

Fields for MQCTLO

Alphabetic list of fields for the MQCTLO structure.

The MQCTLO structure contains the following fields; the fields are described in alphabetical order:

ConnectionArea (MQPTR)

Control options structure - ConnectionArea field

This is a field that is available for the callback function to use.

The queue manager makes no decisions based on the contents of this field and it is passed unchanged to the ConnectionArea field in the MQCTLO structure, which is a parameter on the MQCTL call.

This field is ignored for all operations other than MQOP_START and MQOP_START_WAIT.

This is an input and output field to the callback function. The initial value of this field is a null pointer or null bytes.

Options (MQLONG)

Control options structure - Options field

Options that control the action of MQCTLO.

MQCTLO_FAIL_IF QUIESCING

Force the MQCTLO call to fail if the queue manager or connection is in the quiescing state.

Specify MQGMO_FAIL_IF QUIESCING, in the MQGMO options passed on the MQCB call, to cause notification to message consumers when they are quiescing.

MQCTLO_THREAD_AFFINITY

This option informs the system that the application requires that all message consumers, for the same connection, are called on the same thread. This thread will be used for all invocations of the consumers until the connection is stopped.

Default option: If you do not need any of the options described, use the following option:

MQCTLO_NONE

Use this value to indicate that no other options have been specified; all

options assume their default values. MQCTLO_NONE is defined to aid program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

This is an input field. The initial value of the *Options* field is MQCTLO_NONE.

Reserved (MQCHAR)

This is a reserved field. The initial value of this field is a blank character.

StrucId (MQCHAR4)

Control options structure - StrucId field

This is the structure identifier; the value must be:

MQCTLO_STRUC_ID

Identifier for Control Options structure.

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

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

Version (MQLONG)

Control options structure - Version field

This is the structure version number; the value must be:

MQCTLO_VERSION_1

Version-1 Control options structure.

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

MQCTLO_CURRENT_VERSION

Current version of Control options structure.

This is always an input field. The initial value of this field is MQCTLO_VERSION_1.

Initial values and language declarations for MQCTLO

Control options structure - Initial values

Table 26. Initial values of fields in MQCTLO

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQCTLO_STRUC_ID	'CTLO'
<i>Version</i>	MQCTLO_VERSION_1	1
<i>Options</i>	MQCTLO_NONE	Nulls
<i>Reserved</i>	Reserved field	
<i>ConnectionArea</i>	None	Null pointer or null bytes

Table 26. Initial values of fields in MQCTLO (continued)

Field name	Name of constant	Value of constant
Notes:		
1. In the C programming language, the macro variable MQCTLO_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:		
MQCTLO MyCTLO = {MQCTLO_DEFAULT};		

C declaration

Control Options structure - C language declaration

```
typedef struct tagMQCTLO MQCTLO;
struct tagMQCTLO {
    MQCHAR4  StrucId;          /* Structure identifier */
    MQLONG   Version;         /* Structure version number */
    MQLONG   Options;         /* Options that control the action of MQCTL */
    MQLONG   Reserved;        /* Reserved field */
    MQCHAR   ConnectionArea; /* Connection work area passed to the function */
};
```

COBOL declaration

```
** MQCTLO structure
10 MQCTLO.
** Structure Identifier
15 MQCTLO-STRUCID                PIC X(4).
** Structure Version
15 MQCTLO-VERSION                PIC S9(9) BINARY.
** Options
15 MQCTLO-OPTIONS                PIC S9(9) BINARY.
** Reserved
15 MQCTLO-RESERVED                PIC S9(9) BINARY.
** ConnectionArea
15 MQCTLO-CONNECTIONAREA        POINTER
```

PL/I declaration

```
dc1
1 MQCTLO based,
3 StrucId          char(4),          /* Structure identifier */
3 Version          fixed bin(31),    /* Structure version */
3 Options          fixed bin(31),    /* Options */
3 Reserved         fixed bin(31),
3 ConnectionArea  pointer;          /* Connection work area */
```

MQDH – Distribution header

The following table summarizes the fields in the structure.

Table 27. Fields in MQDH

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>StrucLength</i>	Length of MQDH structure plus following records	StrucLength
<i>Encoding</i>	Numeric encoding of data that follows array of MQPMR records	Encoding

Table 27. Fields in MQDH (continued)

Field	Description	Topic
<i>CodedCharSetId</i>	Character set identifier of data that follows array of MQPMR records	CodedCharSetId
<i>Format</i>	Format name of data that follows array of MQPMR records	Format
<i>Flags</i>	General flags	Flags
<i>PutMsgRecFields</i>	Flags indicating which MQPMR fields are present	PutMsgRecFields
<i>RecsPresent</i>	Number of object records present	RecsPresent
<i>ObjectRecOffset</i>	Offset of first object record from start of MQDH	ObjectRecOffset
<i>PutMsgRecOffset</i>	Offset of first put-message record from start of MQDH	PutMsgRecOffset

Overview for MQDH

Availability: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

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

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

Applications that want to put messages to distribution lists must not use this structure. Instead, they must use the MQOD structure to define the destinations in the distribution list, and the MQPMO structure to specify message properties or receive information about the messages sent to the individual destinations.

Format name: MQFMT_DIST_HEADER.

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

Set the character set and encoding of the MQDH into the *CodedCharSetId* and *Encoding* fields in:

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

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

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

Depending on the destinations, the queue manager can generate more than one such message, and place it 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 can fail the MQPUT or MQPUT1 call with reason code MQRC_DH_ERROR.

You can store messages on a queue in distribution-list form only if you have defined the queue as being able to support distribution list messages (see the *DistLists* queue attribute described in “Attributes for queues” on page 615). 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 for MQDH

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

CodedCharSetId (MQLONG)

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

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

MQCCSI_INHERIT

Inherit character-set identifier of this structure.

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

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

You cannot use MQCCSI_INHERIT if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

This value is supported in the following environments: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG)

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

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

The initial value of this field is 0.

Flags (MQLONG)

You can specify the following flag:

MQDHF_NEW_MSG_IDS

Generate a new message identifier for each destination in the distribution list. Set this only when there are no put-message records present, or when the records are present but they do not contain the *MsgId* field.

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

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

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

If no flags are needed, specify the following:

MQDHF_NONE

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

The initial value of this field is MQDHF_NONE.

Format (MQCHAR8)

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

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

The initial value of this field is MQFMT_NONE.

ObjectRecOffset (MQLONG)

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

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

The initial value of this field is 0.

PutMsgRecFields (MQLONG)

You can specify none or more of the following flags:

MQPMRF_MSG_ID

Message-identifier field is present.

MQPMRF_CORREL_ID

Correlation-identifier field is present.

MQPMRF_GROUP_ID

Group-identifier field is present.

MQPMRF_FEEDBACK

Feedback field is present.

MQPMRF_ACCOUNTING_TOKEN

Accounting-token field is present.

If no MQPMR fields are present, specify the following:

MQPMRF_NONE

No put-message record fields are present. MQPMRF_NONE is defined to aid program documentation. It is not intended that this constant be used with any other, but as its value is zero, such use cannot be detected.

The initial value of this field is MQPMRF_NONE.

PutMsgRecOffset (MQLONG)

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

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

The initial value of this field is 0.

RecsPresent (MQLONG)

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

The initial value of this field is 0.

StrucId (MQCHAR4)

The value must be:

MQDH_STRUC_ID

Identifier for distribution header structure.

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

The initial value of this field is MQDH_STRUC_ID.

StrucLength (MQLONG)

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 MQPMPR records. The data occurs in the following sequence:

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

The arrays of MQOR and MQPMPR records are addressed by offsets contained within the MQDH structure. If these offsets result in unused bytes between one or more of the MQDH structure, the arrays of records, and the message data, those unused bytes must be included in the value of *StrucLength*, but the content of those bytes is not preserved by the queue manager. It is valid for the array of MQPMPR records to precede the array of MQOR records.

The initial value of this field is 0.

Version (MQLONG)

The value must be:

MQDH_VERSION_1

Version number for distribution header structure.

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

MQDH_CURRENT_VERSION

Current version of distribution header structure.

The initial value of this field is MQDH_VERSION_1.

Initial values and language declarations for MQDH

Table 28. Initial values of fields in MQDH for MQDH

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

Table 28. Initial values of fields in MQDH for MQDH (continued)

Field name	Name of constant	Value of constant
Notes:		
1. The symbol <code>b</code> represents a single blank character.		
2. In the C programming language, the macro variable <code>MQDH_DEFAULT</code> contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:		
<pre>MQDH MyDH = {MQDH_DEFAULT};</pre>		

C declaration

```
typedef struct tagMQDH MQDH;
struct tagMQDH {
    MQCHAR4  StrucId;          /* Structure identifier */
    MQLONG   Version;         /* Structure version number */
    MQLONG   StrucLength;     /* Length of MQDH structure plus following
                               MQOR and MQPMR records */
    MQLONG   Encoding;       /* Numeric encoding of data that follows
                               the MQOR and MQPMR records */
    MQLONG   CodedCharSetId; /* Character set identifier of data that
                               follows the MQOR and MQPMR records */
    MQCHAR8  Format;         /* Format name of data that follows the
                               MQOR and MQPMR records */
    MQLONG   Flags;         /* General flags */
    MQLONG   PutMsgRecFields; /* Flags indicating which MQPMR fields are
                               present */
    MQLONG   RecsPresent;    /* Number of MQOR records present */
    MQLONG   ObjectRecOffset; /* Offset of first MQOR record from start
                               of MQDH */
    MQLONG   PutMsgRecOffset; /* Offset of first MQPMR record from start
                               of MQDH */
};
```

COBOL declaration

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

PL/I declaration

```

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

```

Visual Basic declaration

```

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

```

MQDLH – Dead-letter header

The following table summarizes the fields in the structure.

Table 29. Fields in MQDLH

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>Reason</i>	Reason message arrived on dead-letter queue	Reason
<i>DestQName</i>	Name of original destination queue	DestQName
<i>DestQMgrName</i>	Name of original destination queue manager	DestQMgrName

Table 29. Fields in MQDLH (continued)

Field	Description	Topic
<i>Encoding</i>	Numeric encoding of data that follows MQDLH	Encoding
<i>CodedCharSetId</i>	Character set identifier of data that follows MQDLH	CodedCharSetId
<i>Format</i>	Format name of data that follows MQDLH	Format
<i>PutApplType</i>	Type of application that put message on dead-letter queue	PutApplType
<i>PutApplName</i>	Name of application that put message on dead-letter queue	PutApplName
<i>PutDate</i>	Date when message was put on dead-letter queue	PutDate
<i>PutTime</i>	Time when message was put on dead-letter queue	PutTime

Overview for MQDLH

Availability: All WebSphere MQ platforms.

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

Format name: MQFMT_DEAD_LETTER_HEADER.

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

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

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

If a message is too long to put on the dead-letter queue, the application must do 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, do this only if it is known that the originator still has a copy of the message; for example, a message received by a message channel agent from a communication channel.

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

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

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

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

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

- Set the *CodedCharSetId*, *Encoding*, and *Format* fields to the values that describe the data that follows the MQDLH structure, usually the values from the original message descriptor.
- Set the context fields *PutApplType*, *PutApplName*, *PutDate*, and *PutTime* 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.
- Set other fields as appropriate.

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

Getting messages from the dead-letter queue: Applications that get messages from the dead-letter queue must verify that the messages begin with an MQDLH structure. The application can determine whether an MQDLH structure is present by examining the *Format* field in the message descriptor MQMD; if the field has the value MQFMT_DEAD_LETTER_HEADER, the message data begins with an

MQDLH structure. Be aware also that messages that applications get from the dead-letter queue might be truncated if they were originally too long for the queue.

Fields for MQDLH

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

CodedCharSetId (MQLONG)

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

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

MQCCSI_INHERIT

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

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

You cannot use MQCCSI_INHERIT if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

This value is supported in the following environments: AIX, HP-UX, z/OS, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

The initial value of this field is MQCCSI_UNDEFINED.

DestQMgrName (MQCHAR48)

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

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

DestQName (MQCHAR48)

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

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

Encoding (MQLONG)

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

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

The initial value of this field is 0.

Format (MQCHAR8)

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

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

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

PutAppName (MQCHAR28)

This is the name of the application that put the message on the dead-letter (undelivered-message) queue.

The format of the name depends on the *PutApplType* field. The format can vary release to release. See the description of the *PutAppName* field in “MQMD – Message descriptor” on page 181.

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

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

PutApplType (MQLONG)

This is the type of application that put the message on the dead-letter (undelivered-message) queue.

This field has the same meaning as the *PutApplType* field in the message descriptor MQMD (see “MQMD – Message descriptor” on page 181 for details).

If the queue manager redirects the message to the dead-letter queue, *PutApplType* has the value MQAT_QMGR.

The initial value of this field is 0.

PutDate (MQCHAR8)

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

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

- YYYYMMDD

where the characters represent:

YYYY year (four numeric digits)

MM month of year (01 through 12)

DD day of month (01 through 31)

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

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

PutTime (MQCHAR8)

This is time when the 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:

- HHMMSSSTH

where the characters represent:

HH hours (00 through 23)

MM minutes (00 through 59)

SS seconds (00 through 59; see note below)

T tenths of a second (0 through 9)

H hundredths of a second (0 through 9)

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

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

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

Reason (MQLONG)

This identifies the reason why the message was placed on the dead-letter queue instead of on the original destination queue. It should be one of the MQFB_* or MQRC_* values (for example, MQRC_Q_FULL). See the description of the *Feedback* field in “MQMD – Message descriptor” on page 181 for details of the common MQFB_* values that can occur.

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

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

MQFB_APPL_CANNOT_BE_STARTED

An application processing a trigger message cannot start the application named in the *AppId* field of the trigger message (see “MQTM – Trigger message” on page 366).

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

MQFB_APPL_TYPE_ERROR

An application processing a trigger message cannot start the application because the *ApplType* field of the trigger message is not valid (see “MQTM – Trigger message” on page 366).

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

MQFB_BIND_OPEN_CLUSRCVR_DEL

The message was on the SYSTEM.CLUSTER.TRANSMIT.QUEUE intended for a cluster queue that was opened with the MQOO_BIND_ON_OPEN option, but the remote cluster-receiver channel to be used to transmit the message to the destination queue was deleted before the message could be sent. Because MQOO_BIND_ON_OPEN was specified, only the channel selected when the queue was opened can be used to transmit the message. As this channel is no longer available, the message is placed on the dead-letter queue.

MQFB_NOT_A_REPOSITORY_MSG

The message is not a repository message.

MQFB_STOPPED_BY_CHAD_EXIT

The message was stopped by channel auto-definition exit.

MQFB_STOPPED_BY_MSG_EXIT

The message was stopped by channel message exit.

MQFB_TM_ERROR

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

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

MQFB_XMIT_Q_MSG_ERROR

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

The initial value of this field is MQRC_NONE.

StrucId (MQCHAR4)

This is the structure identifier. The value must be:

MQDLH_STRUC_ID

Identifier for dead-letter header structure.

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

The initial value of this field is MQDLH_STRUC_ID.

Version (MQLONG)

This is the structure version number. The value must be:

MQDLH_VERSION_1

Version number for dead-letter header structure.

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

MQDLH_CURRENT_VERSION

Current version of dead-letter header structure.

The initial value of this field is MQDLH_VERSION_1.

Initial values and language declarations for MQDLH

Table 30. Initial values of fields in MQDLH for MQDLH

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

Notes:

1. The symbol `b` represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQDLH_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

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

C declaration

```
typedef struct tagMQDLH MQDLH;
struct tagMQDLH {
    MQCHAR4  StrucId;          /* Structure identifier */
    MQLONG   Version;         /* Structure version number */
    MQLONG   Reason;          /* Reason message arrived on dead-letter
                             (undelivered-message) queue */
    MQCHAR48 DestQName;       /* Name of original destination queue */
    MQCHAR48 DestQMgrName;    /* Name of original destination queue
                             manager */
    MQLONG   Encoding;        /* Numeric encoding of data that follows
                             MQDLH */
};
```

```

MQLONG    CodedCharSetId; /* Character set identifier of data that
                        follows MQDLH */
MQCHAR8   Format;         /* Format name of data that follows
                        MQDLH */
MQLONG    PutApplType;   /* Type of application that put message on
                        dead-letter (undelivered-message)
                        queue */
MQCHAR28  PutApplName;   /* Name of application that put message on
                        dead-letter (undelivered-message)
                        queue */
MQCHAR8   PutDate;       /* Date when message was put on dead-letter
                        (undelivered-message) queue */
MQCHAR8   PutTime;       /* Time when message was put on the
                        dead-letter (undelivered-message)
                        queue */
};

```

COBOL declaration

```

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

```

PL/I declaration

```

dcl
1 MQDLH based,
3 StrucId      char(4),      /* Structure identifier */
3 Version      fixed bin(31), /* Structure version number */
3 Reason       fixed bin(31), /* Reason message arrived on
                        dead-letter (undelivered-message)
                        queue */
3 DestQName    char(48),     /* Name of original destination
                        queue */
3 DestQMgrName char(48),     /* Name of original destination queue
                        manager */
3 Encoding     fixed bin(31), /* Numeric encoding of data that
                        follows MQDLH */
3 CodedCharSetId fixed bin(31), /* Character set identifier of data
                        that follows MQDLH */
3 Format        char(8),      /* Format name of data that follows

```

```

MQDLH */
3 PutAppIType    fixed bin(31), /* Type of application that put
                    message on dead-letter
                    (undelivered-message) queue */
3 PutAppIName    char(28),    /* Name of application that put
                    message on dead-letter
                    (undelivered-message) queue */
3 PutDate        char(8),    /* Date when message was put on
                    dead-letter (undelivered-message)
                    queue */
3 PutTime        char(8);    /* Time when message was put on the
                    dead-letter (undelivered-message)
                    queue */

```

System/390 assembler declaration

```

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

```

Visual Basic declaration

```

Type MQDLH
  StrucId      As String*4  'Structure identifier'
  Version      As Long      'Structure version number'
  Reason       As Long      'Reason message arrived on dead-letter'
                '(undelivered-message) queue'
  DestQName    As String*48 'Name of original destination queue'
  DestQMgrName As String*48 'Name of original destination queue'
                'manager'
  Encoding     As Long      'Numeric encoding of data that follows'
                'MQDLH'
  CodedCharSetId As Long    'Character set identifier of data that'
                'follows MQDLH'
  Format       As String*8  'Format name of data that follows MQDLH'
  PutAppIType As Long      'Type of application that put message on'
                'dead-letter (undelivered-message) queue'
  PutAppIName As String*28 'Name of application that put message on'
                'dead-letter (undelivered-message) queue'
  PutDate      As String*8  'Date when message was put on dead-letter'
                '(undelivered-message) queue'
  PutTime      As String*8  'Time when message was put on the'
                'dead-letter (undelivered-message) queue'
End Type

```

MQDMHO – Delete message handle options

The following table summarizes the fields in the structure.

Table 31. Fields in MQDMHO

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>Options</i>	Options	Options

Overview for MQDMHO

Availability: All WebSphere MQ systems and WebSphere MQ clients.

Purpose: The MQDMHO structure allows applications to specify options that control how message handles are deleted. The structure is an input parameter on the MQDLTMH call.

Character set and encoding: Data in MQDMHO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields for MQDMHO

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

Options (MQLONG)

The value must be:

MQDMHO_NONE

No options specified.

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

StrucId (MQCHAR4)

This is the structure identifier; the value must be:

MQDMHO_STRUC_ID

Identifier for delete message handle options structure.

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

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

Version (MQLONG)

This is the structure version number; the value must be:

MQDMHO_VERSION_1

Version-1 delete message handle options structure.

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

MQDMHO_CURRENT_VERSION

Current version of delete message handle options structure.

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

MQDMHO_VERSION_1.

Initial values and language declarations for MQDMHO

Table 32. Initial values of fields in MQDMHO

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQDMHO_STRUC_ID	'DMHO'
<i>Version</i>	MQDMHO_VERSION_1	1
<i>Options</i>	MQDMHO_NONE	0
Notes:		
<p>1. In the C programming language, the macro variable MQDMHO_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:</p> <pre>MQDMHO MyDMHO = {MQDMHO_DEFAULT};</pre>		

C declaration

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

COBOL declaration

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

PL/I declaration

```
dcl
  1 MQDMHO based,
    3 StrucId      char(4),      /* Structure identifier */
    3 Version      fixed bin(31), /* Structure version number */
    3 Options      fixed bin(31), /* Options that control the action of MQDLTMH */
```

System/390 assembler declaration

```
MQDMHO          DSECT
MQDMHO_STRUCID  DS   CL4   Structure identifier
MQDMHO_VERSION  DS   F     Structure version number
MQDMHO_OPTIONS  DS   F     Options that control the action of
*                MQDLTMH
MQDMHO_LENGTH   EQU  *-MQDMHO
MQDMHO_AREA     DS   CL(MQDMHO_LENGTH)
```

MQDMPO – Delete message property options

The following table summarizes the fields in the structure. MQDMPO structure - delete message property options

Table 33. Fields in MQDMPO

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>Options</i>	Options controlling the action of MQDMPO	Options

Overview for MQDMPO

Availability: All WebSphere MQ systems and WebSphere MQ clients.

Purpose: The MQDMPO structure allows applications to specify options that control how properties of messages are deleted. The structure is an input parameter on the MQDLTMP call.

Character set and encoding: Data in MQDMPO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields for MQDMPO

Delete message property options structure - fields

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

Options (MQLONG)

Delete message property options structure - Options field

Location options: The following options relate to the relative location of the property compared to the property cursor.

MQDMPO_DEL_FIRST

Deletes the first property that matches the specified name.

MQDMPO_DEL_NEXT

Deletes the next property that matches the specified name, continuing the search from the property cursor. If this is the first MQDLTMP call for the specified name, the first property that matches the specified name is deleted.

If the property under the cursor has been deleted, MQINQMP deletes the next matching property following the one that has been deleted.

If a property is added that matches the specified name while iteration is in progress, the property might be deleted during the completion of the iteration. The property will be deleted once the iteration is restarted with MQDMPO_DEL_FIRST.

MQDMPO_DEL_PROP_UNDER_CURSOR

Deletes the property pointed to by the property cursor; that is the property that was last inquired using either the MQIMPO_INQ_FIRST or the MQIMPO_INQ_NEXT option.

The property cursor is reset when the message handle is reused, or when the message handle is specified in the *MsgHandle* field of the MQGMO or MQPMO structure on an MQGET or MQPUT call respectively.

If this option is used when the property cursor has not yet been established or, if the property pointer to by the property cursor has already been deleted, the call fails with completion code MQCC_FAILED and reason MQRC_PROPERTY_NOT_AVAILABLE.

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

MQDPMO_NONE

No options specified.

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

StrucId (MQCHAR4)

Delete message property options structure - StrucId field

This is the structure identifier. The value must be:

MQDMPO_STRUC_ID

Identifier for delete message property options structure.

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

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

Version (MQLONG)

Delete message property options structure - Version field

This is the structure version number. The value must be:

MQDMPO_VERSION_1

Version number for delete message property options structure.

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

MQDMPO_CURRENT_VERSION

Current version of delete message property options structure.

This is always an input field. The initial value of this field is MQDMPO_VERSION_1.

Initial values and language declarations for MQDMPO

Delete message property options structure - Initial values

Table 34. Initial values of fields in MQDMPO

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQDMPO_STRUC_ID	'DMPO'
<i>Version</i>	MQDMPO_VERSION_1	1

Table 34. Initial values of fields in MQDPMO (continued)

Field name	Name of constant	Value of constant
<i>Options</i>	Options that control the action of MQDLTMP	MQDPMO_NONE
Notes:		
<p>1. In the C programming language, the macro variable MQDPMO_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:</p> <pre>MQDPMO MyDPMO = {MQDPMO_DEFAULT};</pre>		

C declaration

Delete message property options structure - C language declaration

```
typedef struct tagMQDPMO MQDPMO;
struct tagMQDPMO {
    MQCHAR4  StrucId;          /* Structure identifier */
    MQLONG   Version;         /* Structure version number */
    MQLONG   Options;        /* Options that control the action of
                             MQDLTMP */
};
```

COBOL declaration

Delete message property options structure - COBOL language declaration

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

PL/I declaration

Delete message property options structure - PL/I language declaration

```
Dcl
  1 MQDPMO based,
  3 StrucId      char(4),      /* Structure identifier */
  3 Version      fixed bin(31), /* Structure version number */
  3 Options      fixed bin(31), /* Options that control the action
                               of MQDLTMP */
```

System/390 assembler declaration

Delete message property options structure - Assembler language declaration

```
MQDPMO          DSECT
MQDPMO_STRUCID  DS   CL4  Structure identifier
MQDPMO_VERSION  DS   F    Structure version number
MQDPMO_OPTIONS  DS   F    Options that control the
*                  action of MQDLTMP
MQDPMO_LENGTH   EQU   *-MQDPMO
MQDPMO_AREA     DS   CL(MQDPMO_LENGTH)
```

MQEPH – Embedded PCF header

The following table summarizes the fields in the structure.

Table 35. Fields in MQEPH

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>StrucLength</i>	Length of MQEPH structure plus the MQCFH and parameter structures that follow it	StrucLength
<i>Encoding</i>	Numeric encoding of data that follows last PCF parameter structure	Encoding
<i>CodedCharSetId</i>	Character set identifier of data that follows last PCF parameter structure	CodedCharSetId
<i>Format</i>	Format name of data that follows last PCF parameter structure	Format
<i>Flags</i>	Flags	Flags
<i>PCFHeader</i>	Programmable command format (PCF) header	PCFHeader

Overview for MQEPH

Availability: All WebSphere MQ platforms.

Purpose: The MQEPH structure describes the additional data that is present in a message when that message is a programmable command format (PCF) message. The *PCFHeader* field defines the PCF parameters that follow this structure and this allows you to follow the PCF message data with other headers.

Format name: MQFMT_EMBEDDED_PCF

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

Set the character set and encoding of the MQEPH into the *CodedCharSetId* and *Encoding* fields in:

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

Usage: You cannot use MQEPH structures to send commands to the command server or any other queue manager PCF-accepting server.

Similarly, the command server or any other queue manager PCF-accepting server do not generate responses or events containing MQEPH structures.

Fields for MQEPH

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

CodedCharSetId (MQLONG)

This is the character set identifier of the data that follows the MQEPH structure and the associated PCF parameters; it does not apply to character data in the MQEPH structure itself.

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG)

This is the numeric encoding of the data that follows the MQEPH structure and the associated PCF parameters; it does not apply to character data in the MQEPH structure itself.

The initial value of this field is 0.

Flags (MQLONG)

The following values are available:

MQEPH_NONE

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

MQEPH_CCSID_EMBEDDED

The character set of the parameters containing character data is specified individually within the CodedCharSetId field in each structure. The character set of the StruId and Format fields is defined by the CodedCharSetId field in the header structure that precedes the MQEPH structure, or by the CodedCharSetId field in the MQMD if the MQEPH is at the start of the message.

The initial value of this field is MQEPH_NONE.

Format (MQCHAR8)

This is the format name of the data that follows the MQEPH structure and the associated PCF parameters.

The initial value of this field is MQFMT_NONE.

PCFHeader (MQCFH)

This is the programmable command format (PCF) header, defining the PCF parameters that follow the MQEPH structure. This enables you to follow the PCF message data with other headers.

The PCF header is initially defined with the the following values:

Table 36. Initial values of fields in MQDH

Field name	Name of constant	Value of constant
Type	MQCFT_NONE	0
StrucLength	MQCFH_STRUC_LENGTH	36
Version	MQCFH_VERSION_3	3
StrucLength	None	0

Table 36. Initial values of fields in MQDH (continued)

Field name	Name of constant	Value of constant
<i>Command</i>	MQCMD_NONE	0
<i>MsgSeqNumber</i>	None	1
<i>Control</i>	MQCFC_LAST	1
<i>CompCode</i>	MQCC_OK	0
<i>Reason</i>	MQRC_NONE	0
<i>ParameterCount</i>	None	0

The application must change the Type from MQCFT_NONE to a valid structure type for the use it is making of the embedded PCF header.

StrucId (MQCHAR4)

The value must be:

MQEPH_STRUC_ID

Identifier for distribution header structure.

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

The initial value of this field is MQEPH_STRUC_ID.

StrucLength (MQLONG)

This is the amount of data preceding the next header structure. It includes:

- The length of the MQEPH header
- The length of all PCF parameters following the header
- Any blank padding following those parameters

StrucLength must be a multiple of 4.

The fixed length part of the structure is defined by MQEPH_STRUC_LENGTH_FIXED.

The initial value of this field is 68.

Version (MQLONG)

The value must be:

MQEPH_VERSION_1

Version number for embedded PCF header structure.

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

MQCFH_VERSION_3

Current version of embedded PCF header structure.

The initial value of this field is MQEPH_VERSION_1.

Initial values and language declarations for MQEPH

Table 37. Initial values of fields in MQEPH for MQEPH

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQEPH_STRUC_ID	'EPHb'
<i>Version</i>	MQEPH_VERSION_1	1
<i>StrucLength</i>	MQEPH_STRUC_LENGTH_FIXED	68
<i>Encoding</i>	None	0
<i>CodedCharSetId</i>	MQCCSI_UNDEFINED	0
<i>Format</i>	MQFMT_NONE	Blanks
<i>Flags</i>	MQEPH_NONE	0
<i>PCFHeader</i>	Names and values as defined in Table 36 on page 122	0

Notes:

1. The symbol b represents a single blank character.
2. In the C programming language, the macro variable MQEPH_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

```
MQEPH MyEPH = {MQEPH_DEFAULT};
```

C declaration

```
typedef struct tagMQEPH MQEPH;
struct tagMQDH {
    MQCHAR4  StrucId;           /* Structure identifier */
    MQLONG   Version;          /* Structure version number */
    MQLONG   StrucLength;      /* Total length of MQEPH including the MQCFH
                               and parameter structures that follow it */
    MQLONG   Encoding;         /* Numeric encoding of data that follows last
                               PCF parameter structure */
    MQLONG   CodedCharSetId;   /* Character set identifier of data that
                               follows last PCF parameter structure */
    MQCHAR8  Format;           /* Format name of data that follows last PCF
                               parameter structure */
    MQLONG   Flags;            /* Flags */
    MQCFH    PCFHeader;        /* Programmable command format header */
};
```

COBOL declaration

```
** MQEPH structure
  10 MQEPH.
**   Structure identifier
    15 MQEPH-STRUCID      PIC X(4).
**   Structure version number
    15 MQEPH-VERSION     PIC S9(9) BINARY.
**   Total length of MQEPH structure including the MQCFH
**   and parameter structures that follow it
    15 MQEPH-STRUCLength PIC S9(9) BINARY.
**   Numeric encoding of data that follows last
**   PCF structure
    15 MQEPH-ENCODING    PIC S9(9) BINARY.
**   Character set identifier of data that
**   follows last PCF parameter structure
    15 MQEPH-CODEDCHARSETID PIC S9(9) BINARY.
**   Format name of data that follows last PCF
**   parameter structure
```



```

15 MQEPH-FORMAT          PIC X(8).
**   Flags
15 MQEPH-FLAGS          PIC S9(9) BINARY.
**   Programmable command format header
15 MQEPH-PCFHEADER.
**   Structure type
20 MQEPH-PCFHEADER-TYPE      PIC S9(9) BINARY.
**   Structure length
20 MQEPH-PCFHEADER-STRULENGTH PIC S9(9) BINARY.
**   Structure version number
20 MQEPH-PCFHEADER-VERSION  PIC S9(9) BINARY.
**   Command identifier
20 MQEPH-PCFHEADER-COMMAND  PIC S9(9) BINARY.
**   Message sequence number
20 MQEPH-PCFHEADER-MSGSEQNUMBER PIC S9(9) BINARY.
**   Control options
20 MQEPH-PCFHEADER-CONTROL  PIC S9(9) BINARY.
**   Completion code
20 MQEPH-PCFHEADER-COMPCODE  PIC S9(9) BINARY.
**   Reason code qualifying completion code
20 MQEPH-PCFHEADER-REASON   PIC S9(9) BINARY.
**   Count of parameter structures
20 MQEPH-PCFHEADER-PARAMETERCOUNT PIC S9(9) BINARY.

```

PL/I declaration

```

dcl
1 MQEPH based,
3 StrucId      char(4),      /* Structure identifier */
3 Version      fixed bin(31), /* Structure version number */
3 StrucLength  fixed bin(31), /* Total Length of MQEPH including the
                               MQCFH and parameter structures that
                               follow it
3 Encoding     fixed bin(31), /* Numeric encoding of data that follows
                               last PCF parameter structure
3 CodedCharSetId fixed bin(31), /* Character set identifier of data that
                               follows last PCF parameter structure
3 Format        char(8),      /* Format name of data that follows last
                               PCF parameter structure */
3 Flags        fixed bin(31), /* Flags */
3 PCFHeader,   /* Programmable command format header
5 Type         fixed bin(31), /* Structure type */
5 StrucLength  fixed bin(31), /* Structure length */
5 Version      fixed bin(31), /* Structure version number */
5 Command      fixed bin(31), /* Command identifier */
5 MsgseqNumber fixed bin(31), /* Message sequence number */
5 Control      fixed bin(31), /* Control options */
5 CompCode     fixed bin(31), /* Completion code */
5 Reason       fixed bin(31), /* Reason code qualifying completion code */
5 ParameterCount fixed bin(31); /* Count of parameter structures */

```

System/390 assembler declaration

```

MQEPH          DSECT
MQEPH_STRUCID  DS   CL4  Structure identifier
MQEPH_VERSION  DS    F   Structure version number
MQEPH_STRULENGTH DS   F   Total length of MQEPH including the
*                               MQCFH and parameter structures that
                               follow it
MQEPH_ENCODING DS    F   Numeric encoding of data that follows
*                               last PCF parameter structure
MQEPH_CODEDCHARSETID DS   F   Character set identifier of data that
*                               follows last PCF parameter structure
MQEPH_FORMAT   DS   CL8  Format name of data that follows last
*                               PCF parameter structure
MQEPH_FLAGS    DS    F   Flags
MQEPH_PCFHEADER DS   0F   Force fullword alignment
MQEPH_PCFHEADER_TYPE DS   F   Structure type

```

```

MQEPH_PCFHEADER_STRUCLNGTH    DS    F    Structure length
MQEPH_PCFHEADER_VERSION      DS    F    Structure version number
MQEPH_PCFHEADER_COMMAND      DS    F    Command identifier
MQEPH_PCFHEADER_MSGSEQNUMBER DS    F    Structure length
MQEPH_PCFHEADER_CONTROL      DS    F    Control options
MQEPH_PCFHEADER_COMPCODE     DS    F    Completion code
MQEPH_PCFHEADER_REASON       DS    F    Reason code qualifying completion code
MQEPH_PCFHEADER_PARAMETER COUNT DS    F    Count of parameter structures
MQEPH_PCFHEADER_LENGTH       EQU    *-MQEPH_PCFHEADER
                                ORG    MQEPH_PCFHEADER
MQEPH_PCFHEADER_AREA         DS    CL(MQEPH_PCFHEADER_LENGTH)
*
MQEPH_LENGTH                  EQU    *-MQEPH
                                ORG    MQEPH
MQEPH_AREA                     DS    CL(MQEPH_LENGTH)

```

Visual Basic declaration

```

Type MQEPH
  StrucId      As String*4 'Structure identifier'
  Version      As Long     'Structure version number'
  StrucLength  As Long     'Total length of MQEPH structure including the MQCFH'
                                'and parameter structures that follow it'
  Encoding     As Long     'Numeric encoding of data that follows last'
                                'PCF parameter structure'
  CodedCharSetId As Long   'Character set identifier of data that'
                                'follows last PCF parameter structure'
  Format       As String*8 'Format name of data that follows last PCF'
                                'parameter structure'
  Flags       As Long     'Flags'
  PCFHeader   As MQCFH   'Programmable command format header'
End Type

```

```
Global MQEPH_DEFAULT As MQEPH
```

MQGMO – Get-message options

The following table summarizes the fields in the structure.

Table 38. Fields in MQGMO

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>Options</i>	Options that control the action of MQGET	MQGMO - Options field
<i>WaitInterval</i>	Wait interval	WaitInterval
<i>Signal1</i>	Signal	Signal1
<i>Signal2</i>	Signal identifier	Signal2
<i>ResolvedQName</i>	Resolved name of destination queue	ResolvedQName
Note: The remaining fields are ignored if <i>Version</i> is less than MQGMO_VERSION_2.		
<i>MatchOptions</i>	Options controlling selection criteria used for MQGET	MatchOptions
<i>GroupStatus</i>	Flag indicating whether message retrieved is in a group	GroupStatus
<i>SegmentStatus</i>	Flag indicating whether message retrieved is a segment of a logical message	SegmentStatus

Table 38. Fields in MQGMO (continued)

Field	Description	Topic
<i>Segmentation</i>	Flag indicating whether further segmentation is allowed for the message retrieved	Segmentation
<i>Reserved1</i>	Reserved	Reserved1
Note: The remaining fields are ignored if <i>Version</i> is less than MQGMO_VERSION_3.		
<i>MsgToken</i>	Message token	MsgToken
<i>ReturnedLength</i>	Length of message data returned (bytes)	ReturnedLength
Note: The remaining fields are ignored if <i>Version</i> is less than MQGMO_VERSION_4.		
<i>Reserved2</i>	Reserved	Reserved2
<i>MsgHandle</i>	The handle to a message that is to be populated with the properties of the message being retrieved from the queue.	MsgHandle

Overview for MQGMO

Availability: All WebSphere MQ platforms.

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

Version: The current version of MQGMO is MQGMO_VERSION_4. Certain fields are available only in certain versions of MQGMO. If you need to port applications between several environments, you must ensure that the version of MQGMO is consistent across all environments. Fields that exist only in particular versions of the structure are identified as such in MQGMO - Get-message options and in the field descriptions.

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

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

Fields for MQGMO

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

GroupStatus (MQCHAR)

This flag indicates whether the message retrieved is in a group.

It has one of the following values:

MQGS_NOT_IN_GROUP

Message is not in a group.

MQGS_MSG_IN_GROUP

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

MQGS_LAST_MSG_IN_GROUP

Message is the last in the group.

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

This is an output field. The initial value of this field is `MQGS_NOT_IN_GROUP`. This field is ignored if *Version* is less than `MQGMO_VERSION_2`.

MatchOptions (MQLONG)

These options allow the application to choose which fields in the *MsgDesc* parameter to use to select the message returned by the MQGET call. The application sets the required options in this field, and then sets the corresponding fields in the *MsgDesc* parameter to the values required for those fields. Only messages that have those values in the MQMD for the message are candidates for retrieval using that *MsgDesc* parameter on the MQGET call. Fields for which the corresponding match option is *not* specified are ignored when selecting the message to be returned. If you specify no selection criteria on the MQGET call (that is, *any* message is acceptable), set *MatchOptions* to `MQMO_NONE`.

- On z/OS, the selection criteria that can be used might be restricted by the type of index used for the queue. See the *IndexType* queue attribute for further details.

If you specify `MQGMO_LOGICAL_ORDER`, only certain messages are eligible for return by the next MQGET call:

- If there is no current group or logical message, only messages that have *MsgSeqNumber* equal to 1 and *Offset* equal to 0 are eligible for return. In this situation, you can use one or more of the following match options to select which of the eligible messages is returned:
 - `MQMO_MATCH_MSG_ID`
 - `MQMO_MATCH_CORREL_ID`
 - `MQMO_MATCH_GROUP_ID`
- If there *is* a current group or logical message, only the next message in the group or next segment in the logical message is eligible for return, and this cannot be altered by specifying `MQMO_*` options.

In both of the above cases, you can specify match options that do not apply, but the value of the relevant field in the *MsgDesc* parameter must match the value of the corresponding field in the message to be returned; the call fails with reason code `MQRC_MATCH_OPTIONS_ERROR` if this condition is not satisfied.

MatchOptions is ignored if you specify either `MQGMO_MSG_UNDER_CURSOR` or `MQGMO_BROWSE_MSG_UNDER_CURSOR`.

You can specify one or more of the following match options:

MQMO_MATCH_MSG_ID

The message to be retrieved must have a message identifier that matches the value of the *MsgId* field in the *MsgDesc* parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the correlation identifier).

If you omit this option, the *MsgId* field in the *MsgDesc* parameter is ignored, and any message identifier will match.

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

MQMO_MATCH_CORREL_ID

The message to be retrieved must have a correlation identifier that matches the value of the *CorrelId* field in the *MsgDesc* parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the message identifier).

If you omit this option, the *CorrelId* field in the *MsgDesc* parameter is ignored, and any correlation identifier will match.

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

MQMO_MATCH_GROUP_ID

The message to be retrieved must have a group identifier that matches the value of the *GroupId* field in the *MsgDesc* parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the correlation identifier).

If you omit this option, the *GroupId* field in the *MsgDesc* parameter is ignored, and any group identifier will match.

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

MQMO_MATCH_MSG_SEQ_NUMBER

The message to be retrieved must have a message sequence number that matches the value of the *MsgSeqNumber* field in the *MsgDesc* parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the group identifier).

If you omit this option, the *MsgSeqNumber* field in the *MsgDesc* parameter is ignored, and any message sequence number will match.

MQMO_MATCH_OFFSET

The message to be retrieved must have an offset that matches the value of the *Offset* field in the *MsgDesc* parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the message sequence number).

If you omit this option is not specified, the *Offset* field in the *MsgDesc* parameter is ignored, and any offset will match.

- This option is not supported on z/OS.

MQMO_MATCH_MSG_TOKEN

The message to be retrieved must have a message token that matches the value of the *MsgToken* field in the MQGMO structure specified on the MQGET call.

You can specify this option for all local queues. If you specify it for a queue that has an *IndexType* of MQIT_MSG_TOKEN (a WLM-managed queue), you can specify no other match options with MQMO_MATCH_MSG_TOKEN.

You cannot specify MQMO_MATCH_MSG_TOKEN with MQGMO_WAIT or MQGMO_SET_SIGNAL. If the application wants to wait for a message to arrive on a queue that has an *IndexType* of MQIT_MSG_TOKEN, specify MQMO_NONE.

If you omit this option, the *MsgToken* field in MQGMO is ignored, and any message token will match.

If you specify none of the options described, you can use the following option:

MQMO_NONE

Use no matches in selecting the message to be returned; all messages on the queue are eligible for retrieval (but subject to control by the MQGMO_ALL_MSGS_AVAILABLE, MQGMO_ALL_SEGMENTS_AVAILABLE, and MQGMO_COMPLETE_MSG options).

MQMO_NONE aids program documentation. It is not intended that this option be used with any other MQMO_* option, but as its value is zero, such use cannot be detected.

This is an input field. The initial value of this field is MQMO_MATCH_MSG_ID with MQMO_MATCH_CORREL_ID. This field is ignored if *Version* is less than MQGMO_VERSION_2.

Note: The initial value of the *MatchOptions* field is defined for compatibility with earlier MQSeries® queue managers. However, when reading a series of messages from a queue without using selection criteria, this initial value requires the application to reset the *MsgId* and *CorrelId* fields to MQMI_NONE and MQCI_NONE prior to each MQGET call. Avoid the need to reset *MsgId* and *CorrelId* by setting *Version* to MQGMO_VERSION_2, and *MatchOptions* to MQMO_NONE.

MsgHandle (MQHMSG)

If the MQGMO_PROPERTIES_AS_Q_DEF option is specified and the *PropertyControl* queue attribute is not set to MQPROP_FORCE_MQRFH2 then this is the handle to a message which will be populated with the properties of the message being retrieved from the queue. The handle is created by an MQCRTMH call. Any properties already associated with the handle will be cleared before retrieving a message.

The following value can also be specified:

MQHM_NONE

No message handle supplied.

No message descriptor is required on the MQGET call if a valid message handle is supplied and used on output to contain the message properties, the message descriptor associated with the message handle is used for input fields.

If a message descriptor is specified on the MQGET call, it always takes precedence over the message descriptor associated with a message handle.

If MQGMO_PROPERTIES_FORCE_MQRFH2 is specified, or the MQGMO_PROPERTIES_AS_Q_DEF is specified and the *PropertyControl* queue attribute is MQPROP_FORCE_MQRFH2 then the call fails with reason code MQRC_MD_ERROR when no message descriptor parameter is specified.

On return from the MQGET call, the properties and message descriptor associated with this message handle are updated to reflect the state of the message retrieved (as well as the message descriptor if one was supplied on the MQGET call). The properties of the message can then be inquired using the MQINQMP call.

Except for message descriptor extensions, when present, a property that can be inquired with the MQINQMP call is not contained in the message data; if the message on the queue contained properties in the message data these are removed from the message data before the data is returned to the application.

If no message handle is provided or Version is less than MQGMO_VERSION_4 then you must supply a valid message descriptor on the MQGET call. Any message properties (except those contained in the message descriptor) are returned in the message data subject to the value of the property options in the MQGMO structure and the *PropertyControl* queue attribute.

This is an always an input field. The initial value of this field is MQHM_NONE. This field is ignored if *Version* is less than MQGMO_VERSION_4.

MsgToken (MQBYTE16)

MsgToken field - MQGMO structure. This field is used by the queue manager to uniquely identify a message.

This is a byte string that is generated by the queue manager to identify a message uniquely on a queue. The message token is generated when the message is first placed on the queue manager, and remains with the message until the message is permanently removed from the queue manager, unless the queue manager is restarted.

When the message is removed from the queue, the *MsgToken* that identified that instance of the message is no longer valid, and is never reused. If the queue manager is restarted, the *MsgToken* that identified a message on the queue before restart might not be valid after restart. However, the *MsgToken* is never reused to identify a different message instance. The *MsgToken* is generated by the queue manager and is not visible to any external application.

When a message is returned by a call to MQGET where a Version 3 or higher MQGMO is supplied, the *MsgToken* identifying the message on the queue is returned in the MQGMO by the queue manager. There is one exception to this: when the message is being removed from the queue outside syncpoint, the queue manager might not return a *MsgToken* because it is not useful to identify the returned message on a subsequent MQGET call. Applications should only use *MsgToken* to refer to the message on subsequent MQGET calls.

If a *MsgToken* is supplied and the *MatchOption* MQMO_MATCH_MSG_TOKEN is specified and neither MQGMO_MSG_UNDER_CURSOR nor MQGMO_BROWSE_MSG_UNDER_CURSOR is specified, only the message identified by that *MsgToken* can be returned. The option is valid on all local queues regardless of INDXTYPE, and on z/OS you must use INDXTYPE(MSGTOKEN) only on Work Load Manager (WLM) queues.

Any other *MatchOptions* specified are checked, and if they do not match, MQRC_NO_MSG_AVAILABLE is returned. If MQGMO_BROWSE_NEXT is coded with MQMO_MATCH_MSG_TOKEN, the message identified by the *MsgToken* is returned only if it is beyond the browse-cursor for the calling handle.

If MQGMO_MSG_UNDER_CURSOR or MQGMO_BROWSE_MSG_UNDER_CURSOR is specified, MQMO_MATCH_MSG_TOKEN is ignored.

MQMO_MATCH_MSG_TOKEN is not valid with the following get message options:

- MQGMO_WAIT
- MQGMO_SET_SIGNAL

For an MQGET call specifying MQMO_MATCH_MSG_TOKEN, an MQGMO of version 3 or later must be supplied to the call, otherwise MQRC_WRONG_GMO_VERSION is returned.

If the *MsgToken* is not valid at this time, MQCC_FAILED with MQRC_NO_MSG_AVAILABLE is returned, unless there is another error.

Options (MQLONG)

These options control the action of MQGET. You can specify none or more of the options described below. If you need more than one the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

Combinations of options that are not valid are noted; all other combinations are valid.

Wait options: The following options relate to waiting for messages to arrive on the queue:

MQGMO_WAIT

The application waits until a suitable message arrives. The maximum time that the application waits is specified in *WaitInterval*.

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

You can use this option with the MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT options.

If several applications are waiting on the same shared queue, the applications that are activated when a suitable message arrives are described below.

Note: In the description below, a *browse* MQGET call is one that specifies one of the browse options, but *not* MQGMO_LOCK; an MQGET call specifying the MQGMO_LOCK option is treated as a *nonbrowse* call.

- If one or more nonbrowse MQGET calls is waiting, but no browse MQGET calls are waiting, one is activated.

- If one or more browse MQGET calls is waiting, but no nonbrowse MQGET calls are waiting, all are activated.
- If one or more nonbrowse MQGET calls, and one or more browse MQGET calls are waiting, one nonbrowse MQGET call is activated, and none, some, or all of the browse MQGET calls. (The number of browse MQGET calls activated cannot be predicted, because it depends on the scheduling considerations of the operating system, and other factors.)

If more than one nonbrowse MQGET call is waiting on the same queue, only one is activated; in this situation the queue manager attempts to give priority to waiting nonbrowse calls in the following order:

1. Specific get-wait requests that can be satisfied only by certain messages, for example, ones with a specific *MsgId* or *CorrelId* (or both).
2. General get-wait requests that can be satisfied by any message.

Note the following points:

- Within the first category, no additional priority is given to more specific get-wait requests, for example those that specify both *MsgId* and *CorrelId*.
- Within either category, it cannot be predicted which application is selected. In particular, the application waiting longest is not necessarily the one selected.
- Path length, and priority-scheduling considerations of the operating system, can mean that a waiting application of lower operating system priority than expected retrieves the message.
- It can also happen that an application that is not waiting retrieves the message in preference to one that is.

On z/OS, the following points apply:

- If you want the application to proceed with other work while waiting for the message to arrive, consider using the signal option (MQGMO_SET_SIGNAL) instead. However the signal option is environment specific; applications that you to port between different environments must not use it.
- If there is more than one MQGET call waiting for the same message, with a mixture of wait and signal options, each waiting call is considered equally. It is an error to specify MQGMO_SET_SIGNAL with MQGMO_WAIT. It is also an error to specify this option with a queue handle for which a signal is outstanding.
- If you specify MQGMO_WAIT or MQGMO_SET_SIGNAL for a queue that has an *IndexType* of MQIT_MSG_TOKEN, no selection criteria are permitted. This means that:
 - If you are using a version-1 MQGMO, set the *MsgId* and *CorrelId* fields in the MQMD specified on the MQGET call to MQMI_NONE and MQCI_NONE respectively.
 - If you are using a version-2 or later MQGMO, set the *MatchOptions* field to MQMO_NONE.

MQGMO_WAIT is ignored if specified with MQGMO_BROWSE_MSG_UNDER_CURSOR or MQGMO_MSG_UNDER_CURSOR; no error is raised.

MQGMO_NO_WAIT

The application does not wait if no suitable message is available. This is

the opposite of the MQGMO_WAIT option, and is defined to aid program documentation. It is the default if neither is specified.

MQGMO_SET_SIGNAL

Use this option with the *Signal1* and *Signal2* fields to allow applications to proceed with other work while waiting for a message to arrive, and also (if suitable operating system facilities are available) to wait for messages arriving on more than one queue.

Note: The MQGMO_SET_SIGNAL option is environment specific; do not use it for applications that you want to port.

If a currently available message satisfies the criteria specified in the message descriptor, or if a parameter error or other synchronous error is detected, the call completes in the same way as if this option had not been specified.

If no message satisfying the criteria specified in the message descriptor is currently available, control returns to the application without waiting for a message to arrive. The output fields in the message descriptor and the output parameters of the MQGET call are not set, other than the *CompCode* and *Reason* parameters (which are set to MQCC_WARNING and MQRC_SIGNAL_REQUEST_ACCEPTED respectively). When a suitable message arrives subsequently, the signal is delivered by posting the ECB.

The caller must then reissue the MQGET call to retrieve the message. The application can wait for this signal, using functions provided by the operating system.

If the operating system provides a multiple wait mechanism, the application can use this technique to wait for a message arriving on any one of several queues.

If a nonzero *WaitInterval* is specified, after this time the signal is delivered. The queue manager can also cancel the wait, in which case the signal is delivered.

If more than one MQGET call has set a signal for the same message, the order in which applications are activated is the same as that described for MQGMO_WAIT.

If there is more than one MQGET call waiting for the same message, with a mixture of wait and signal options, each waiting call is considered equally.

Under certain conditions the MQGET call can retrieve a message, *and* a signal resulting from the arrival of the same message can be delivered. When a signal is delivered, an application must be prepared for no message to be available.

A queue handle can have no more than one signal request outstanding.

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

- MQGMO_UNLOCK
- MQGMO_WAIT

This option is supported on z/OS only.

MQGMO_FAIL_IF QUIESCING

Force the MQGET call to fail if the queue manager is in the quiescing state.

On z/OS, this option also forces the MQGET call to fail if the connection (for a CICS or IMS application) is in the quiescing state.

If this option is specified with MQGMO_WAIT or MQGMO_SET_SIGNAL, and the wait or signal is outstanding at the time the queue manager enters the quiescing state:

- The wait is canceled and the call returns completion code MQCC_FAILED with reason code MQRC_Q_MGR QUIESCING or MQRC_CONNECTION QUIESCING.
- The signal is canceled with an environment-specific signal completion code.

On z/OS, the signal completes with event completion code MQEC_Q_MGR QUIESCING or MQEC_CONNECTION QUIESCING.

If MQGMO_FAIL_IF QUIESCING is not specified and the queue manager or connection enters the quiescing state, the wait or signal is not canceled.

Syncpoint options: The following options relate to the participation of the MQGET call within a unit of work:

MQGMO_SYNCPOINT

The request is to operate within the normal unit-of-work protocols. The message is marked as being unavailable to other applications, but it is deleted from the queue only when the unit of work is committed. The message is made available again if the unit of work is backed out.

If neither this option nor MQGMO_NO_SYNCPOINT is specified, the inclusion of the get request in unit-of-work protocols is determined by the environment:

- On z/OS, the get request is within a unit of work.
- In all other environments, the get request is not within a unit of work.

Because of these differences, an application that you want to port must not allow this option to default; specify MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT explicitly.

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

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT
- MQGMO_LOCK
- MQGMO_NO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_UNLOCK

MQGMO_SYNCPOINT_IF_PERSISTENT

The request is to operate within the normal unit-of-work protocols, but *only* if the message retrieved is persistent. A persistent message has the value MQPER_PERSISTENT in the *Persistence* field in MQMD.

- If the message is persistent, the queue manager processes the call as though the application had specified MQGMO_SYNCPOINT (see above for details).
- If the message is not persistent, the queue manager processes the call as though the application had specified MQGMO_NO_SYNCPOINT (see below for details).

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

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT
- MQGMO_COMPLETE_MSG
- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_NO_SYNCPOINT
- MQGMO_SYNCPOINT
- MQGMO_UNLOCK

This option is supported in the following environments: AIX, HP-UX, z/OS, i5/OS, Solaris, and Linux, plus WebSphere MQ clients connected to these systems.

MQGMO_NO_SYNCPOINT

The request is to operate outside the normal unit-of-work protocols. The message is deleted from the queue immediately (unless this is a browse request). The message cannot be made available again by backing out the unit of work.

This option is assumed if you specify MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT.

If you specify neither this option nor MQGMO_SYNCPOINT, the inclusion of the get request in unit-of-work protocols is determined by the environment:

- On z/OS, the get request is within a unit of work.
- In all other environments, the get request is not within a unit of work.

Because of these differences, an application that you want to port must not allow this option to default; specify either MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT explicitly.

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

- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT

MQGMO_MARK_SKIP_BACKOUT

Back out a unit of work without reinstating on the queue the message that was marked with this option.

When an application requests the backout of a unit of work containing a get request, a message that was retrieved using this option is not restored to its previous state. (Other resource updates, however, are still backed out.) Instead, the message is treated as if it had been retrieved by a get request *without* this option, in a new unit of work started by the backout request.

This is useful if the application retrieves a message, but only after some resources have been changed does it become apparent that the unit of work cannot complete successfully. If you omit this option, backing out the unit of work reinstates the message on the queue, so that the same sequence of events occurs when the message is next retrieved. However, if you specify this option on the original MQGET call, backing out the unit of work backs out the updates to the other resources, but treats the message as if it had been retrieved under a new unit of work. The application can

perform appropriate error handling (such as sending a report message to the sender of the original message, or placing the original message on the dead-letter queue), and then commit the new unit of work. Committing the new unit of work removes the message permanently from the original queue.

MQGMO_MARK_SKIP_BACKOUT marks a single physical message. If the message belongs to a message group, the other messages in the group are not marked. Similarly, if the marked message is a segment of a logical message, the other segments in the logical message are not marked. Any message in a group can be marked, but if messages are retrieved using MQGMO_LOGICAL_ORDER, it is advantageous to mark the *first* message in the group. When the unit of work is backed out, the first (marked) message is moved to the new unit of work, while the second and later messages in the group are reinstated on the queue. However, that group is no longer eligible for retrieval by an application using MQGMO_LOGICAL_ORDER, because the first message in the group is no longer on the queue. This prevents a second instance of the application inadvertently processing messages in that group. However, the first instance of the application can retrieve the second and later messages into the new unit of work using the MQGMO_LOGICAL_ORDER option as normal.

Occasionally you might need to back out the new unit of work (for example, because the dead-letter queue is full and the message must not be discarded). Backing out the new unit of work reinstates the message on the original queue, which prevents the message being lost. However, in this situation processing cannot continue. After backing out the new unit of work, the application must inform the operator or administrator that there is an unrecoverable error, and then terminate.

This option has an effect only if the unit of work containing the get request is terminated by the application backing it out. (Such requests use calls or commands that depend on the environment.) This option has no effect if the unit of work containing the get request is backed out for any other reason (for example, because the transaction or system abends). In this situation, any message retrieved using this option is reinstated on the queue in the same way as messages retrieved without this option.

Note:

1. If you have not applied IMS APAR PN60855, an IMS MPP or BMP application that backs out a unit of work containing a message retrieved with the MQGMO_MARK_SKIP_BACKOUT option must issue an MQ call (any MQ call will do) before committing or backing out the new unit of work.
2. A CICS application that backs out a unit of work containing a message retrieved with the MQGMO_MARK_SKIP_BACKOUT option must issue an MQ call (any MQ call will do) before committing or backing out the new unit of work.

Within a unit of work, there can be only one get request marked as skipping backout, as well as none or several unmarked get requests.

If this option is specified, MQGMO_SYNCPOINT must also be specified. MQGMO_MARK_SKIP_BACKOUT is not valid with any of the following options:

- MQGMO_BROWSE_FIRST

- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT
- MQGMO_LOCK
- MQGMO_NO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_UNLOCK

This option is supported only on z/OS.

Browse options: The following options relate to browsing messages on the queue:

MQGMO_BROWSE_FIRST

When a queue is opened with the MQOO_BROWSE option, a browse cursor is established, positioned logically before the first message on the queue. You can then use MQGET calls specifying the MQGMO_BROWSE_FIRST, MQGMO_BROWSE_NEXT, or MQGMO_BROWSE_MSG_UNDER_CURSOR option to retrieve messages from the queue nondestructively. The browse cursor marks the position, within the messages on the queue, from which the next MQGET call with MQGMO_BROWSE_NEXT searches for a suitable message.

An MQGET call with MQGMO_BROWSE_FIRST ignores the previous position of the browse cursor. The first message on the queue that satisfies the conditions specified in the message descriptor is retrieved. The message remains on the queue, and the browse cursor is positioned on this message.

After this call, the browse cursor is positioned on the message that has been returned. If the message is removed from the queue before the next MQGET call with MQGMO_BROWSE_NEXT is issued, the browse cursor remains at the position in the queue that the message occupied, even though that position is now empty.

The MQGMO_MSG_UNDER_CURSOR option can subsequently be used with a nonbrowse MQGET call if required, to remove the message from the queue.

The browse cursor is not moved by a nonbrowse MQGET call using the same *Hobj* handle. Nor is it moved by a browse MQGET call that returns a completion code of MQCC_FAILED, or a reason code of MQRC_TRUNCATED_MSG_FAILED.

Specify the MQGMO_LOCK option with this option, to lock the message that is browsed.

You can specify MQGMO_BROWSE_FIRST with any valid combination of the MQGMO_* and MQMO_* options that control the processing of messages in groups and segments of logical messages.

If you specify MQGMO_LOGICAL_ORDER, the messages are browsed in logical order. If you omit that option, the messages are browsed in physical order. When you specify MQGMO_BROWSE_FIRST, you can switch between logical order and physical order, but subsequent MQGET calls using MQGMO_BROWSE_NEXT must browse the queue in the same order as the most-recent call that specified MQGMO_BROWSE_FIRST for the queue handle.

The group and segment information that the queue manager retains for MQGET calls that browse messages on the queue is separate from the group and segment information that the queue manager retains for

MQGET calls that remove messages from the queue. When you specify MQGMO_BROWSE_FIRST, the queue manager ignores the group and segment information for browsing, and scans the queue as though there were no current group and no current logical message. If the MQGET call is successful (completion code MQCC_OK or MQCC_WARNING), the group and segment information for browsing is set to that of the message returned; if the call fails, the group and segment information remains the same as it was before the call.

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

- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT
- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_MSG_UNDER_CURSOR
- MQGMO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_UNLOCK

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

MQGMO_BROWSE_NEXT

Advance the browse cursor to the next message on the queue that satisfies the selection criteria specified on the MQGET call. The message is returned to the application, but remains on the queue.

After a queue has been opened for browse, the first browse call using the handle has the same effect whether it specifies the MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT option.

If the message is removed from the queue before the next MQGET call with MQGMO_BROWSE_NEXT is issued, the browse cursor logically remains at the position in the queue that the message occupied, even though that position is now empty.

Messages are stored on the queue in one of two ways:

- FIFO within priority (MQMDS_PRIORITY), or
- FIFO *regardless* of priority (MQMDS_FIFO)

The *MsgDeliverySequence* queue attribute indicates which method applies (see “Attributes for queues” on page 615 for details).

If the queue has a *MsgDeliverySequence* of MQMDS_PRIORITY, and a message arrives on the queue that is of a higher priority than the one currently pointed to by the browse cursor, that message is not found during the current sweep of the queue using MQGMO_BROWSE_NEXT. It can be found only after the browse cursor has been reset with MQGMO_BROWSE_FIRST (or by reopening the queue).

The MQGMO_MSG_UNDER_CURSOR option can subsequently be used with a nonbrowse MQGET call if required, to remove the message from the queue.

The browse cursor is not moved by nonbrowse MQGET calls using the same *Hobj* handle.

Specify the MQGMO_LOCK option with this option to lock the message that is browsed.

You can specify MQGMO_BROWSE_NEXT with any valid combination of the MQGMO_* and MQMO_* options that control the processing of messages in groups and segments of logical messages.

If you specify MQGMO_LOGICAL_ORDER, the messages are browsed in logical order. If you omit that option, the messages are browsed in physical order. When you specify MQGMO_BROWSE_FIRST, you can switch between logical order and physical order, but subsequent MQGET calls using MQGMO_BROWSE_NEXT must browse the queue in the same order as the most-recent call that specified MQGMO_BROWSE_FIRST for the queue handle. The call fails with reason code MQRC_INCONSISTENT_BROWSE if this condition is not satisfied.

Note: Take special care when using an MQGET call to browse *beyond the end* of a message group (or logical message not in a group) when MQGMO_LOGICAL_ORDER is not specified. For example, if the last message in the group *precedes* the first message in the group on the queue, using MQGMO_BROWSE_NEXT to browse beyond the end of the group, specifying MQMO_MATCH_MSG_SEQ_NUMBER with *MsgSeqNumber* set to 1 (to find the first message of the next group) returns the first message in the group already browsed. This can happen immediately, or a number of MQGET calls later (if there are intervening groups).

To avoid the possibility of an infinite loop, open the queue *twice* for browse:

- Use the first handle to browse only the first message in each group.
- Use the second handle to browse only the messages within a specific group.
- Use the MQMO_* options to move the second browse cursor to the position of the first browse cursor, before browsing the messages in the group.
- Do not use MQGMO_BROWSE_NEXT to browse beyond the end of a group.

The group and segment information that the queue manager retains for MQGET calls that browse messages on the queue is separate from the group and segment information that it retains for MQGET calls that remove messages from the queue.

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

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_MSG_UNDER_CURSOR
- MQGMO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_UNLOCK

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

MQGMO_BROWSE_MSG_UNDER_CURSOR

Retrieve the message pointed to by the browse cursor nondestructively, regardless of the MQMO_* options specified in the *MatchOptions* field in MQGMO.

The message pointed to by the browse cursor is the one that was last retrieved using either the MQGMO_BROWSE_FIRST or the MQGMO_BROWSE_NEXT option. The call fails if neither of these calls has been issued for this queue since it was opened, or if the message that was under the browse cursor has since been retrieved destructively.

The position of the browse cursor is not changed by this call.

The MQGMO_MSG_UNDER_CURSOR option can subsequently be used with a nonbrowse MQGET call if required, to remove the message from the queue.

The browse cursor is not moved by a nonbrowse MQGET call using the same *Hobj* handle. Nor is it moved by a browse MQGET call that returns a completion code of MQCC_FAILED, or a reason code of MQRC_TRUNCATED_MSG_FAILED.

If MQGMO_BROWSE_MSG_UNDER_CURSOR is specified *with* MQGMO_LOCK:

- If there is already a message locked, it must be the one under the cursor, so that is returned *without* unlocking and relocking it; the message remains locked.
- If there is no locked message, the message under the browse cursor (if there is one) is locked and returned to the application; if there is no message under the browse cursor the call fails.

If MQGMO_BROWSE_MSG_UNDER_CURSOR is specified *without* MQGMO_LOCK:

- If there is already a message locked, it must be the one under the cursor. This message is returned to the application *and then unlocked*. Because the message is now unlocked, there is no guarantee that it can be browsed again, or retrieved destructively (it can be retrieved destructively by another application getting messages from the queue).
- If there is no locked message, the message under the browse cursor (if there is one) is returned to the application; if there is no message under the browse cursor the call fails.

If MQGMO_COMPLETE_MSG is specified with MQGMO_BROWSE_MSG_UNDER_CURSOR, the browse cursor must identify a message whose *Offset* field in MQMD is zero. If this condition is not satisfied, the call fails with reason code MQRC_INVALID_MSG_UNDER_CURSOR.

The group and segment information that the queue manager retains for MQGET calls that browse messages on the queue is separate from the group and segment information that it retains for MQGET calls that remove messages from the queue.

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

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_NEXT
- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_MSG_UNDER_CURSOR
- MQGMO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_UNLOCK

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

MQGMO_MSG_UNDER_CURSOR

Retrieve the message pointed to by the browse cursor, regardless of the MQMO_* options specified in the *MatchOptions* field in MQGMO. The message is removed from the queue.

The message pointed to by the browse cursor is the one that was last retrieved using either the MQGMO_BROWSE_FIRST or the MQGMO_BROWSE_NEXT option.

If MQGMO_COMPLETE_MSG is specified with MQGMO_MSG_UNDER_CURSOR, the browse cursor must identify a message whose *Offset* field in MQMD is zero. If this condition is not satisfied, the call fails with reason code MQRC_INVALID_MSG_UNDER_CURSOR.

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

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT
- MQGMO_UNLOCK

It is also an error if the queue was not opened both for browse and for input. If the browse cursor is not currently pointing to a retrievable message, an error is returned by the MQGET call.

MQGMO_MARK_BROWSE_HANDLE

After a successful call to MQGET that specifies this option, the message that is returned, or that is identified by the *MsgToken* that is returned, is considered, by the object handle used in the call, to be marked. The message is not removed from the queue.

This option is valid only if one of the following options is also specified:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT

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

- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE
- MQGMO_COMPLETE_MSG
- MQGMO_LOCK
- MQGMO_LOGICAL_ORDER
- MQGMO_UNLOCK

The message remains in this state until one of the following occurs:

- The object handle concerned is closed, either normally or otherwise.
- The message is unmarked for this handle by a call to MQGET with the option MQGMO_UNMARK_BROWSE_HANDLE.
- The message is returned from a call to destructive MQGET, which completes with MQCC_OK or MQCC_WARNING. This is true even if the MQGET is subsequently rolled-back.
- The message expires.

MQGMO_MARK_BROWSE_CO_OP

After a successful call to MQGET that specifies this option, the message that is returned, or that is identified by the *MsgToken* that is returned, is

considered by any object handle that is part of the cooperating set of handles, to be marked for the cooperating set of handles.

The message is not considered to be marked for each individual handle in the set at a handle level. That is, this option is not exactly equivalent to each handle in the cooperating set browsing the message with `MQGMO_MARK_BROWSE_HANDLE`, but all handles in the set can determine that the message has been marked. The message is not removed from the queue.

This option is valid only if the object handle used was returned by a successful call to `MQOPEN` that specified the `MQOO_CO_OP` option and one of the following `MQGMO` options is also specified:

- `MQGMO_BROWSE_FIRST`
- `MQGMO_BROWSE_MSG_UNDER_CURSOR`
- `MQGMO_BROWSE_NEXT`

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

- `MQGMO_ALL_MSGS_AVAILABLE`
- `MQGMO_ALL_SEGMENTS_AVAILABLE`
- `MQGMO_COMPLETE_MSG`
- `MQGMO_LOCK`
- `MQGMO_LOGICAL_ORDER`
- `MQGMO_UNLOCK`

If the message is already considered to be in this state, that is, with `MQGMO_MARK_BROWSE_CO_OP`, and the option `MQGMO_UNMARKED_BROWSE_MSG` is not specified, the call fails with `MQCC_FAILED` and reason code `MQRC_MSG_MARKED_BROWSE_CO_OP`.

The message remains in this state until one of the following occurs:

- All object handles in the cooperating set are closed.
- The message is unmarked for cooperating browsers by a call to `MQGET` with the option `MQGMO_UNMARK_BROWSE_CO_OP`.
- The message is automatically unmarked by the queue manager.
- The message is returned from a call to a non-browse `MQGET`. This is true even if the `MQGET` is subsequently rolled-back.
- The message expires.

`MQGMO_UNMARKED_BROWSE_MSG`

A call to `MQGET` that specifies this option does not return a message that is considered, by the handle that is used, to be marked. If the handle that is used was returned by a successful call to `MQOPEN`, with the option `MQOO_CO_OP`, the call also does not return a message that is considered to be marked for the cooperating set of handles with `MQGMO_MARK_BROWSE_CO_OP`.

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

- `MQGMO_ALL_MSGS_AVAILABLE`
- `MQGMO_ALL_SEGMENTS_AVAILABLE`
- `MQGMO_COMPLETE_MSG`
- `MQGMO_LOCK`
- `MQGMO_LOGICAL_ORDER`

- MQGMO_UNLOCK

MQGMO_UNMARK_BROWSE_CO_OP

After a call to MQGET that specifies this option, the message located is no longer considered by all other open handles in the set of cooperating handles to be marked for the cooperating set. The message is still considered to be marked at handle level by any open handle that considered it to be marked at handle level before this call.

This call is valid only using an open handle that was returned from a successful call to MQOPEN with the option MQOO_CO_OP, and succeeds even if the message is not considered to be marked by the cooperating set of handles.

This option is not valid on a non-browse MQGET call, or with any of the following options:

- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE
- MQGMO_COMPLETE_MSG
- MQGMO_LOCK
- MQGMO_LOGICAL_ORDER
- MQGMO_MARK_BROWSE_CO_OP
- MQGMO_UNLOCK
- MQGMO_UNMARKED_BROWSE_MSG

MQGMO_UNMARK_BROWSE_HANDLE

After a call to MQGET that specifies this option, the message located is no longer considered to be marked by this handle.

This call succeeds even if the message is not marked for this handle.

This option is not valid on a non-browse MQGET call, or with any of the following options:

- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE
- MQGMO_COMPLETE_MSG
- MQGMO_LOCK
- MQGMO_LOGICAL_ORDER
- MQGMO_MARK_BROWSE_CO_OP
- MQGMO_UNLOCK
- MQGMO_UNMARKED_BROWSE_MSG

Lock options: The following options relate to locking messages on the queue:

MQGMO_LOCK

Lock the message that is browsed, so that the message becomes invisible to any other handle open for the queue. The option can be specified only if one of the following options is also specified:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_NEXT
- MQGMO_BROWSE_MSG_UNDER_CURSOR

Only one message can be locked for each queue handle, but this can be a logical message or a physical message:

- If you specify MQGMO_COMPLETE_MSG, all the message segments that comprise the logical message are locked to the queue handle (provided that they are all present on the queue and available for retrieval).
- If you omit MQGMO_COMPLETE_MSG, only a single physical message is locked to the queue handle. If this message happens to be a segment of a logical message, the locked segment prevents other applications using MQGMO_COMPLETE_MSG to retrieve or browse the logical message.

The locked message is always the one under the browse cursor, and the message can be removed from the queue by a later MQGET call that specifies the MQGMO_MSG_UNDER_CURSOR option. Other MQGET calls using the queue handle can also remove the message (for example, a call that specifies the message identifier of the locked message).

If the call returns completion code MQCC_FAILED, or MQCC_WARNING with reason code MQRC_TRUNCATED_MSG_FAILED, no message is locked.

If the application does not remove the message from the queue, the lock is released by:

- Issuing another MQGET call for this handle, specifying either MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT (with or without MQGMO_LOCK); the message is unlocked if the call completes with MQCC_OK or MQCC_WARNING, but remains locked if the call completes with MQCC_FAILED. However, the following exceptions apply:
 - The message *is not* unlocked if MQCC_WARNING is returned with MQRC_TRUNCATED_MSG_FAILED.
 - The message *is* unlocked if MQCC_FAILED is returned with MQRC_NO_MSG_AVAILABLE.

If you also specify MQGMO_LOCK, the message returned is locked. If you omit MQGMO_LOCK, there is no locked message after the call.

If you specify MQGMO_WAIT, and no message is immediately available, the unlock on the original message occurs before the start of the wait (providing the call is otherwise free from error).

- Issuing another MQGET call for this handle, with MQGMO_BROWSE_MSG_UNDER_CURSOR (without MQGMO_LOCK); the message is unlocked if the call completes with MQCC_OK or MQCC_WARNING, but remains locked if the call completes with MQCC_FAILED. However, the following exception applies:
 - The message *is not* unlocked if MQCC_WARNING is returned with MQRC_TRUNCATED_MSG_FAILED.
- Issuing another MQGET call for this handle with MQGMO_UNLOCK.
- Issuing an MQCLOSE call for this handle (either explicitly, or implicitly by the application ending).

No special open option is required to specify this option, other than MQOO_BROWSE, which is needed to specify the accompanying browse option.

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

- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_SYNCPOINT

- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_UNLOCK

MQGMO_UNLOCK

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

The *MsgDesc*, *BufferLength*, *Buffer*, and *DataLength* parameters are not checked or altered if you specify MQGMO_UNLOCK. No message is returned in *Buffer*.

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

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

- MQGMO_NO_WAIT
- MQGMO_NO_SYNCPOINT

Both of these options are assumed whether specified or not.

Message-data options: The following options relate to the processing of the message data when the message is read from the queue:

MQGMO_ACCEPT_TRUNCATED_MSG

If the message buffer is too small to hold the complete message, allow 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 that:

- When browsing messages, the browse cursor is advanced to the returned message.
- When removing messages, the returned message is removed from the queue.
- Reason code MQRC_TRUNCATED_MSG_ACCEPTED is returned if no other error occurs.

Without this option, the buffer is still filled with as much of the message as it can hold, a warning completion code is issued, but processing is not completed. This means that:

- When browsing messages, the browse cursor is not advanced.
- When removing messages, the message is not removed from the queue.
- Reason code MQRC_TRUNCATED_MSG_FAILED is returned if no other error occurs.

MQGMO_CONVERT

This option converts the application data in the message to conform to the *CodedCharSetId* and *Encoding* values specified in the *MsgDesc* parameter on the MQGET call, before the data is copied to the *Buffer* parameter.

The *Format* field specified when the message was put is assumed by the conversion process to identify the nature of the data in the message. The message data is converted by the queue manager for built-in formats, and by a user-written exit for other formats. See Chapter 8, “Data conversion,” on page 677 for details of the data-conversion exit.

- If conversion is successful, the *CodedCharSetId* and *Encoding* fields specified in the *MsgDesc* parameter are unchanged on return from the MQGET call.

- If conversion fails (but the MQGET call otherwise completes without error), the message data is returned unconverted, and the *CodedCharSetId* and *Encoding* fields in *MsgDesc* are set to the values for the unconverted message. The completion code is MQCC_WARNING in this case.

In either case, these fields describe the character-set identifier and encoding of the message data that is returned in the *Buffer* parameter.

See the *Format* field described in “MQMD – Message descriptor” on page 181 for a list of format names for which the queue manager performs the conversion.

Group and segment options: The following options relate to the processing of messages in groups and segments of logical messages. Before the option descriptions, here are some definitions of important terms:

Physical message

This is the smallest unit of information that can be placed on or removed from a queue; it often corresponds to the information specified or retrieved on a single MQPUT, MQPUT1, or MQGET call. Every physical message has its own message descriptor (MQMD). Generally, physical messages are distinguished by differing values for the message identifier (*MsgId* field in MQMD), although this is not enforced by the queue manager.

Logical message

This is a single unit of application information. In the absence of system constraints, a logical message is the same as a physical message. But where logical messages are extremely large, system constraints might make it advisable or necessary to split a logical message into two or more physical messages, called *segments*.

A logical message that has been segmented consists of two or more physical messages that have the same nonnull group identifier (*GroupId* field in MQMD), and the same message sequence number (*MsgSeqNumber* field in MQMD). The segments are distinguished by differing values for the segment offset (*Offset* field in MQMD), which gives the offset of the data in the physical message from the start of the data in the logical message. Because each segment is a physical message, the segments in a logical message usually have different message identifiers.

A logical message that has not been segmented, but for which segmentation has been permitted by the sending application, also has a nonnull group identifier, although in this case there is only one physical message with that group identifier if the logical message does not belong to a message group. Logical messages for which segmentation has been inhibited by the sending application have a null group identifier (MQGI_NONE), unless the logical message belongs to a message group.

Message group

This is a set of one or more logical messages that have the same nonnull group identifier. The logical messages in the group are distinguished by different values for the message sequence number, which is an integer in the range 1 through n, where n is the number of logical messages in the group. If one or more of the logical messages is segmented, there will be more than n physical messages in the group.

MQGMO_LOGICAL_ORDER

This option controls the order in which messages are returned by *successive*

MQGET calls for the queue handle. The option must be specified on each of those calls in order to have an effect.

If MQGMO_LOGICAL_ORDER is specified for successive MQGET calls for the queue handle, messages in groups are returned in the order given by their message sequence numbers, and segments of logical messages are returned in the order given by their segment offsets. This order might be different from the order in which those messages and segments occur on the queue.

Note: Specifying MQGMO_LOGICAL_ORDER has no adverse consequences on messages that do not belong to groups and that are not segments. In effect, such messages are treated as though each belonged to a message group consisting of only one message. Thus it is perfectly safe to specify MQGMO_LOGICAL_ORDER when retrieving messages from queues that might 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 before each MQGET call. Specifically, it means that the application does not need to set the *GroupId*, *MsgSeqNumber*, and *Offset* fields in MQMD. However, the application must set the MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT option correctly on each call.

When the queue is opened, there is no current message group and no current logical message. A message group becomes the current message group when a message that has the MQMF_MSG_IN_GROUP flag is returned by the MQGET call. With MQGMO_LOGICAL_ORDER specified on successive calls, that group remains the current group until a message is returned that has:

- MQMF_LAST_MSG_IN_GROUP without MQMF_SEGMENT (that is, the last logical message in the group is not segmented), or
- MQMF_LAST_MSG_IN_GROUP with MQMF_LAST_SEGMENT (that is, the message returned is the last segment of the last logical message in the group).

When such a message is returned, the message group is terminated, and on successful completion of that MQGET call there is no longer a current group. In a similar way, a logical message becomes the current logical message when a message that has the MQMF_SEGMENT flag is returned by the MQGET call, and that logical message is terminated when the message that has the MQMF_LAST_SEGMENT flag is returned.

If no selection criteria are specified, successive MQGET calls return (in the correct order) the messages for the first message group on the queue, then the messages for the second message group, and so on, until there are no more messages available. It is possible to select the particular message groups returned by specifying one or more of the following options in the *MatchOptions* field:

- MQMO_MATCH_MSG_ID
- MQMO_MATCH_CORREL_ID

- MQMO_MATCH_GROUP_ID

However, these options are effective only when there is no current message group or logical message; see the *MatchOptions* field described in “MQGMO – Get-message options” on page 126 for further details.

Table 39 shows the values of the *MsgId*, *CorrelId*, *GroupId*, *MsgSeqNumber*, and *Offset* fields that the queue manager looks for when attempting to find a message to return on the MQGET call. This applies both to removing messages from the queue, and browsing messages on the queue. In the table, Either means Yes or No:

LOG ORD

Indicates whether the MQGMO_LOGICAL_ORDER option is specified on the call.

Cur grp

Indicates whether a current message group exists prior to the call.

Cur log msg

Indicates whether a current logical message exists prior to the call.

Other columns

Show the values that the queue manager looks for. Previous denotes the value returned for the field in the previous message for the queue handle.

Table 39. MQGET options relating to messages in groups and segments of logical messages

Options you specify	Group and log-msg status prior to call		Values the queue manager looks for				
	Cur grp	Cur log msg	<i>MsgId</i>	<i>CorrelId</i>	<i>GroupId</i>	<i>MsgSeqNumber</i>	<i>Offset</i>
LOG ORD							
Yes	No	No	Controlled by <i>MatchOptions</i>	Controlled by <i>MatchOptions</i>	Controlled by <i>MatchOptions</i>	1	0
Yes	No	Yes	Any message identifier	Any correlation identifier	Previous group identifier	1	Previous offset + previous segment length
Yes	Yes	No	Any message identifier	Any correlation identifier	Previous group identifier	Previous sequence number + 1	0
Yes	Yes	Yes	Any message identifier	Any correlation identifier	Previous group identifier	Previous sequence number	Previous offset + previous segment length
No	Either	Either	Controlled by <i>MatchOptions</i>	Controlled by <i>MatchOptions</i>	Controlled by <i>MatchOptions</i>	Controlled by <i>MatchOptions</i>	Controlled by <i>MatchOptions</i>

When multiple message groups are present on the queue and eligible for return, the groups are returned in the order determined by the position on the queue of the first segment of the first logical message in each group (that is, the physical messages that have message sequence numbers of 1, and offsets of 0, determine the order in which eligible groups are returned).

The MQGMO_LOGICAL_ORDER option affects units of work as follows:

- If the first logical message or segment in a group is retrieved within a unit of work, all 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, and the same queue handle is used, none of the other logical messages and segments in the group can be retrieved within a unit of work.

If these conditions are not satisfied, the MQGET call fails with reason code MQRC_INCONSISTENT_UOW.

When MQGMO_LOGICAL_ORDER is specified, the MQGMO supplied on the MQGET call must not be less than MQGMO_VERSION_2, and the MQMD must not be less than MQMD_VERSION_2. If this condition is not satisfied, the call fails with reason code MQRC_WRONG_GMO_VERSION or MQRC_WRONG_MD_VERSION, as appropriate.

If MQGMO_LOGICAL_ORDER is *not* specified for successive MQGET calls for the queue handle, messages are returned without regard for whether they belong to message groups, or whether they are segments of logical messages. This means that messages or segments from a particular group or logical message might be returned out of order, or intermingled with messages or segments from other groups or logical messages, or with messages that are not in groups and are not segments. In this situation, the particular messages that are returned by successive MQGET calls is controlled by the MQMO_* options specified on those calls (see the *MatchOptions* field described in “MQGMO – Get-message options” on page 126 for details of these options).

This is the technique that can be used to restart a message group or logical message in the middle, after a system failure has occurred. When the system restarts, the application can set the *GroupId*, *MsgSeqNumber*, *Offset*, and *MatchOptions* fields to the appropriate values, and then issue the MQGET call with MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT set, but *without* specifying MQGMO_LOGICAL_ORDER. If this call is successful, the queue manager retains the group and segment information, and subsequent MQGET calls using that queue handle can specify MQGMO_LOGICAL_ORDER as normal.

The group and segment information that the queue manager retains for the MQGET call is separate from the group and segment information that it retains for the MQPUT call. In addition, the queue manager retains separate information for:

- MQGET calls that remove messages from the queue.
- MQGET calls that browse messages on the queue.

For any given queue handle, the application can mix MQGET calls that specify MQGMO_LOGICAL_ORDER with MQGET calls that do not. However, note the following points:

- If you omit MQGMO_LOGICAL_ORDER, each successful MQGET call causes the queue manager to set the saved group and segment information to the values corresponding to the message returned; this replaces the existing group and segment information retained by the queue manager for the queue handle. Only the information appropriate to the action of the call (browse or remove) is modified.
- If you omit MQGMO_LOGICAL_ORDER, the call does not fail if there is a current message group or logical message; the call might succeed with an MQCC_WARNING completion code. Table 40 on page 151 shows the

various cases that can arise. In these cases, if the completion code is not MQCC_OK, the reason code is one of the following (as appropriate):

- MQRC_INCOMPLETE_GROUP
- MQRC_INCOMPLETE_MSG
- MQRC_INCONSISTENT_UOW

Note: The queue manager does not check the group and segment information when browsing a queue, or when closing a queue that was opened for browse but not input; in those cases the completion code is always MQCC_OK (assuming no other errors).

Table 40. Outcome when MQGET or MQCLOSE call is not consistent with group and segment information

Current call is	Previous call was MQGET with MQGMO_LOGICAL_ORDER	Previous call was MQGET without MQGMO_LOGICAL_ORDER
MQGET with MQGMO_LOGICAL_ORDER	MQCC_FAILED	MQCC_FAILED
MQGET without MQGMO_LOGICAL_ORDER	MQCC_WARNING	MQCC_OK
MQCLOSE with an unterminated group or logical message	MQCC_WARNING	MQCC_OK

Applications that want to retrieve messages and segments in logical order are recommended to specify MQGMO_LOGICAL_ORDER, as this is the simplest option to use. This option relieves the application of the need to manage the group and segment information, because the queue manager manages that information. However, specialized applications might need more control than that provided by the MQGMO_LOGICAL_ORDER option, and this can be achieved by not specifying that option. The application must then ensure that the *MsgId*, *CorrelId*, *GroupId*, *MsgSeqNumber*, and *Offset* fields in MQMD, and the MQMO_* options in *MatchOptions* in MQGMO, are set correctly, before 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, must *not* specify MQGMO_LOGICAL_ORDER. In a complex network with multiple paths between sending and receiving queue managers, the physical messages might arrive out of order. By specifying neither MQGMO_LOGICAL_ORDER, nor the corresponding MQPMO_LOGICAL_ORDER on the MQPUT call, the forwarding application can retrieve and forward each physical message as soon as it arrives, without having to wait for the next one in logical order to arrive.

You can specify MQGMO_LOGICAL_ORDER with any of the other MQGMO_* options, and with various of the MQMO_* options in appropriate circumstances (see above).

- On z/OS, this option is supported for private and shared queues, but the queue must have an index type of MQIT_GROUP_ID. For shared queues, the CFSTRUCT object that the queue maps to must be at CFLEVEL(3) or CFLEVEL(4).
- On AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems, this option is supported for all local queues.

MQGMO_COMPLETE_MSG

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 want to receive individual segments must always specify MQGMO_COMPLETE_MSG.

To use this option, the application must provide a buffer that is big enough to accommodate the complete message, or specify the MQGMO_ACCEPT_TRUNCATED_MSG option.

If the queue contains segmented messages with some of the segments missing (perhaps because they have been delayed in the network and have not yet arrived), specifying MQGMO_COMPLETE_MSG prevents the retrieval of segments belonging to incomplete logical messages. However, those message segments still contribute to the value of the *CurrentQDepth* queue attribute; this means that there might 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 during 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 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 exists, the queue manager cannot reassemble. If the message does not require reassembly, the call can still succeed. But if the message requires reassembly, the call fails with reason code MQRC_UOW_NOT_AVAILABLE.

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

Each physical message that is a segment has its own message descriptor. For the segments constituting a single logical message, most of the fields in the message descriptor are the same for all segments in the logical message; usually it is only the *MsgId*, *Offset*, and *MsgFlags* fields that differ between segments in the logical message. However, if a segment is placed on a dead-letter queue at an intermediate queue manager, the DLQ handler retrieves the message specifying the MQGMO_CONVERT option, and this can result in the character set or encoding of the segment being changed. If the DLQ handler successfully sends the segment on its way, the

segment might have a character set or encoding that differs from the other segments in the logical message when the segment arrives at the destination queue manager.

A logical message consisting of segments in which the *CodedCharSetId* and *Encoding* fields differ cannot be reassembled by the queue manager into a single logical message. Instead, the queue manager reassembles and returns the first few consecutive segments at the start of the logical message that have the same character-set identifiers and encodings, and the MQGET call completes with completion code MQCC_WARNING and reason code MQRC_INCONSISTENT_CCSDS or MQRC_INCONSISTENT_ENCODINGS, as appropriate. This happens regardless of whether MQGMO_CONVERT is specified. To retrieve the remaining segments, the application must reissue the MQGET call without the MQGMO_COMPLETE_MSG option, retrieving the segments one by one. MQGMO_LOGICAL_ORDER can be used to retrieve the remaining segments in order.

An application that puts segments can also set other fields in the message descriptor to values that differ between segments. However, there is no advantage in doing this if the receiving application uses MQGMO_COMPLETE_MSG to retrieve the logical message. When the queue manager reassembles a logical message, it returns in the message descriptor the values from the message descriptor for the *first* segment; the only exception is the *MsgFlags* field, which the queue manager sets to indicate that the reassembled message is the only segment.

If MQGMO_COMPLETE_MSG is specified for a report message, the queue manager performs special processing. The queue manager checks the queue to see if all the report messages of that report type relating to the different segments in the logical message are present on the queue. If they are, they can be retrieved as a single message by specifying MQGMO_COMPLETE_MSG. For this to be possible, either the report messages must be generated by a queue manager or MCA which supports segmentation, or the originating application must request at least 100 bytes of message data (that is, the appropriate MQRO_*_WITH_DATA or MQRO_*_WITH_FULL_DATA options must be specified). If less than the full amount of application data is present for a segment, the missing bytes are replaced by nulls in the report message returned.

If MQGMO_COMPLETE_MSG is specified with MQGMO_MSG_UNDER_CURSOR or MQGMO_BROWSE_MSG_UNDER_CURSOR, the browse cursor must be positioned on a message whose *Offset* field in MQMD has a value of 0. If this condition is not satisfied, the call fails with reason code MQRC_INVALID_MSG_UNDER_CURSOR.

MQGMO_COMPLETE_MSG implies MQGMO_ALL_SEGMENTS_AVAILABLE, which need not therefore be specified.

MQGMO_COMPLETE_MSG can be specified with any of the other MQGMO_* options apart from MQGMO_SYNCPOINT_IF_PERSISTENT, and with any of the MQMO_* options apart from MQMO_MATCH_OFFSET.

- On z/OS, this option is supported for private and shared queues, but the queue must have an index type of MQIT_GROUP_ID. For shared queues, the CFSTRUCT object that the queue map to must be at CFLEVEL(3) or CFLEVEL(4).

- On AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems, this option is supported for all local queues.

MQGMO_ALL_MSGS_AVAILABLE

Messages in a group become available for retrieval only when *all* messages in the group are available. If the queue contains message groups with some of the messages missing (perhaps because they have been delayed in the network and have not yet arrived), specifying MQGMO_ALL_MSGS_AVAILABLE prevents retrieval of messages belonging to incomplete groups. However, those messages still contribute to the value of the *CurrentQDepth* queue attribute; this means that there may be no retrievable message groups, even though *CurrentQDepth* is greater than zero. If there are no other messages that are retrievable, reason code MQRC_NO_MSG_AVAILABLE is returned after the specified wait interval (if any) has expired.

The processing of MQGMO_ALL_MSGS_AVAILABLE depends on whether MQGMO_LOGICAL_ORDER is also specified:

- If both options are specified, MQGMO_ALL_MSGS_AVAILABLE has an effect *only* when there is no current group or logical message. If there *is* a current group or logical message, MQGMO_ALL_MSGS_AVAILABLE is ignored. This means that MQGMO_ALL_MSGS_AVAILABLE can remain on when processing messages in logical order.
- If MQGMO_ALL_MSGS_AVAILABLE is specified without MQGMO_LOGICAL_ORDER, MQGMO_ALL_MSGS_AVAILABLE *always* has an effect. This means that the option must be turned off after the first message in the group has been removed from the queue, in order to be able to remove the remaining messages in the group.

Successful completion of an MQGET call specifying MQGMO_ALL_MSGS_AVAILABLE means that at the time that the MQGET call was issued, all the messages in the group were on the queue. However, be aware that other applications can still remove messages from the group (the group is not locked to the application that retrieves the first message in the group).

If you omit this option, messages belonging to groups can be retrieved even when the group is incomplete.

MQGMO_ALL_MSGS_AVAILABLE implies MQGMO_ALL_SEGMENTS_AVAILABLE, which need not therefore be specified.

MQGMO_ALL_MSGS_AVAILABLE can be specified with any of the other MQGMO_* options, and with any of the MQMO_* options.

- On z/OS, this option is supported for private and shared queues, but the queue must have an index type of MQIT_GROUP_ID. For shared queues, the CFSTRUCT object that the queue map to must be at CFLEVEL(3) or CFLEVEL(4).
- On AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems, this option is supported for all local queues.

MQGMO_ALL_SEGMENTS_AVAILABLE

Segments in a logical message become available for retrieval only when *all* segments in the logical message are available. If the queue contains segmented messages with some of the segments missing (perhaps because

they have been delayed in the network and have not yet arrived), specifying `MQGMO_ALL_SEGMENTS_AVAILABLE` prevents retrieval of segments belonging to incomplete logical messages. However, those segments still contribute to the value of the *CurrentQDepth* queue attribute; this means that there might be no retrievable logical messages, even though *CurrentQDepth* is greater than zero. If there are no other messages that are retrievable, reason code `MQRC_NO_MSG_AVAILABLE` is returned after the specified wait interval (if any) has expired.

The processing of `MQGMO_ALL_SEGMENTS_AVAILABLE` depends on whether `MQGMO_LOGICAL_ORDER` is also specified:

- If both options are specified, `MQGMO_ALL_SEGMENTS_AVAILABLE` has an effect *only* when there is no current logical message. If there *is* a current logical message, `MQGMO_ALL_SEGMENTS_AVAILABLE` is ignored. This means that `MQGMO_ALL_SEGMENTS_AVAILABLE` can remain on when processing messages in logical order.
- If `MQGMO_ALL_SEGMENTS_AVAILABLE` is specified without `MQGMO_LOGICAL_ORDER`, `MQGMO_ALL_SEGMENTS_AVAILABLE` *always* has an effect. This means that the option must be turned off after the first segment in the logical message has been removed from the queue, in order to be able to remove the remaining segments in the logical message.

If this option is not specified, message segments can be retrieved even when the logical message is incomplete.

While both `MQGMO_COMPLETE_MSG` and `MQGMO_ALL_SEGMENTS_AVAILABLE` require all segments to be available before any of them can be retrieved, the former returns the complete message, whereas the latter allows the segments to be retrieved one by one.

If `MQGMO_ALL_SEGMENTS_AVAILABLE` is specified for a report message, the queue manager checks the queue to see if there is at least one report message for each of the segments that comprise the complete logical message. If there is, the `MQGMO_ALL_SEGMENTS_AVAILABLE` condition is satisfied. However, the queue manager does not check the *type* of the report messages present, and so there might be a mixture of report types in the report messages relating to the segments of the logical message. As a result, the success of `MQGMO_ALL_SEGMENTS_AVAILABLE` does not imply that `MQGMO_COMPLETE_MSG` will succeed. If there *is* a mixture of report types present for the segments of a particular logical message, those report messages must be retrieved one by one.

You can specify `MQGMO_ALL_SEGMENTS_AVAILABLE` with any of the other `MQGMO_*` options, and with any of the `MQMO_*` options.

- On z/OS, this option is supported for private and shared queues, but the queue must have an index type of `MQIT_GROUP_ID`. For shared queues, the `CFSTRUCT` object that the queue map to must be at `CFLEVEL(3)` or `CFLEVEL(4)`.
- On AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems, this option is supported for all local queues.

Property options: The following options relate to the properties of the message:

`MQGMO_PROPERTIES_AS_Q_DEF`

Properties of the message, except those contained in the message descriptor (or extension) should be represented as defined by the *PropertyControl* queue attribute. If a *MsgHandle* is provided this option is ignored and the properties of the message are available via the *MsgHandle*, unless the value of the *PropertyControl* queue attribute is MQPROP_FORCE_MQRFH2.

This is the default action if no property options are specified.

MQGMO_PROPERTIES_IN_HANDLE

Properties of the message should be made available via the *MsgHandle*. If no message handle is provided the call fails with reason MQRC_HMSG_ERROR.

MQGMO_NO_PROPERTIES

No properties of the message, except those contained in the message descriptor (or extension) will be retrieved. If a *MsgHandle* is provided it will be ignored.

MQGMO_PROPERTIES_FORCE_MQRFH2

Properties of the message, except those contained in the message descriptor (or extension) should be represented using MQRFH2 headers. This provides backward compatibility for applications which are expecting to retrieve properties but are unable to be changed to use message handles. If a *MsgHandle* is provided it is ignored.

MQGMO_PROPERTIES_COMPATIBILITY

If the message contains a property with a prefix of "mcd.", "jms.", "usr.", or "mqext.", all message properties are delivered to the application in an MQRFH2 header. Otherwise all properties of the message, except those contained in the message descriptor (or extension), are discarded and are no longer accessible to the application.

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

MQGMO_NONE

Use this value to indicate that no other options have been specified; all options assume their default values. MQGMO_NONE aids program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

The initial value of the *Options* field is MQGMO_NO_WAIT plus MQGMO_PROPERTIES_AS_Q_DEF.

Reserved1 (MQCHAR)

This is a reserved field. The initial value of this field is a blank character. This field is ignored if *Version* is less than MQGMO_VERSION_2.

Reserved2 (MQLONG)

This is a reserved field. The initial value of this field is 0. This field is ignored if *Version* is less than MQPMO_VERSION_4.

ResolvedQName (MQCHAR48)

This is an output field that the queue manager sets to the local name of the queue from which the message was retrieved, as defined to the local queue manager. This is different from the name used to open the queue if:

- An alias queue was opened (in which case, the name of the local queue to which the alias resolved is returned), or
- A model queue was opened (in which case, the name of the dynamic local queue is returned).

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

ReturnedLength (MQLONG)

This is an output field that the queue manager sets to the length in bytes of the message data returned by the `MQGET` call in the *Buffer* parameter. If the queue manager does not support this capability, *ReturnedLength* is set to the value `MQRL_UNDEFINED`.

When messages are converted between encodings or character sets, the message data can sometimes change size. On return from the `MQGET` call:

- If *ReturnedLength* is *not* `MQRL_UNDEFINED`, the number of bytes of message data returned is given by *ReturnedLength*.
- If *ReturnedLength* has the value `MQRL_UNDEFINED`, the number of bytes of message data returned is usually given by the smaller of *BufferLength* and *DataLength*, but can be *less than* this if the `MQGET` call completes with reason code `MQRC_TRUNCATED_MSG_ACCEPTED`. If this happens, the insignificant bytes in the *Buffer* parameter are set to nulls.

The following special value is defined:

MQRL_UNDEFINED

Length of returned data not defined.

On z/OS, the value returned for the *ReturnedLength* field is always `MQRL_UNDEFINED`.

The initial value of this field is `MQRL_UNDEFINED`. This field is ignored if *Version* is less than `MQGMO_VERSION_3`.

Segmentation (MQCHAR)

This is a flag that indicates whether further segmentation is allowed for the message retrieved. It has one of the following values:

MQSEG_INHIBITED

Segmentation not allowed.

MQSEG_ALLOWED

Segmentation allowed.

On z/OS, the queue manager always sets this field to `MQSEG_INHIBITED`.

This is an output field. The initial value of this field is `MQSEG_INHIBITED`. This field is ignored if *Version* is less than `MQGMO_VERSION_2`.

SegmentStatus (MQCHAR)

This is a flag that indicates whether the message retrieved is a segment of a logical message. It has one of the following values:

MQSS_NOT_A_SEGMENT

Message is not a segment.

MQSS_SEGMENT

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

MQSS_LAST_SEGMENT

Message is the last segment of the logical message.

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

On z/OS, the queue manager always sets this field to MQSS_NOT_A_SEGMENT.

This is an output field. The initial value of this field is MQSS_NOT_A_SEGMENT. This field is ignored if *Version* is less than MQGMO_VERSION_2.

Signal1 (MQLONG)

This is an input field that is used only in conjunction with the MQGMO_SET_SIGNAL option; it identifies a signal that is to be delivered when a message is available.

Note: The data type and usage of this field are determined by the environment; for this reason, applications that you want to port between different environments must not use signals.

- On z/OS, this field must contain the address of an Event Control Block (ECB). The ECB must be cleared by the application before the MQGET call is issued. The storage containing the ECB must not be freed until the queue is closed. The ECB is posted by the queue manager with one of the signal completion codes described below. These completion codes are set in bits 2 through 31 of the ECB, the area defined in the z/OS mapping macro IHAECB as being for a user completion code.
- In all other environments, this is a reserved field; its value is not significant.

The signal completion codes are:

MQEC_MSG_ARRIVED

A suitable message has arrived on the queue. This message has not been reserved for the caller; a second MQGET request must be issued, but another application might retrieve the message before the second request is made.

MQEC_WAIT_INTERVAL_EXPIRED

The specified *WaitInterval* has expired without a suitable message arriving.

MQEC_WAIT_CANCELED

The wait was canceled for an indeterminate reason (such as the queue manager terminating or the queue being disabled). Reissue the request if you want further diagnosis.

MQEC_Q_MGR QUIESCING

The wait was canceled because the queue manager has entered the quiescing state (MQGMO_FAIL_IF QUIESCING was specified on the MQGET call).

MQEC_CONNECTION QUIESCING

The wait was canceled because the connection has entered the quiescing state (MQGMO_FAIL_IF QUIESCING was specified on the MQGET call).

The initial value of this field is determined by the environment:

- On z/OS, the initial value is the null pointer.
- In all other environments, the initial value is 0.

Signal2 (MQLONG)

This is an input field that is used only in conjunction with the MQGMO_SET_SIGNAL option. It is a reserved field; its value is not significant.

The initial value of this field is 0.

StrucId (MQCHAR4)

This is the structure identifier. The value must be:

MQGMO_STRUC_ID

Identifier for get-message options structure.

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

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

Version (MQLONG)

Version is the structure version number.

The value must be one of the following:

MQGMO_VERSION_1

Version-1 get-message options structure.

This version is supported in all environments.

MQGMO_VERSION_2

Version-2 get-message options structure.

This version is supported in all environments.

MQGMO_VERSION_3

Version-3 get-message options structure.

This version is supported in all environments.

MQGMO_VERSION_4

Version-4 get-message options structure.

This version is supported in all environments.

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

MQGMO_CURRENT_VERSION

Current version of get-message options structure.

This is always an input field. The initial value of this field is MQGMO_VERSION_1.

WaitInterval (MQLONG)

This is the approximate time, expressed in milliseconds, that the MQGET call waits for a suitable message to arrive (that is, a message satisfying the selection criteria specified in the *MsgDesc* parameter of the MQGET call; see the *MsgId* field described in “MQMD – Message descriptor” on page 181 for more details). If no suitable message has arrived after this time has elapsed, the call completes with MQCC_FAILED and reason code MQRC_NO_MSG_AVAILABLE.

On z/OS, the period of time that the MQGET call actually waits is affected by system loading and work-scheduling considerations, and can vary between the value specified for *WaitInterval* and approximately 250 milliseconds greater than *WaitInterval*.

WaitInterval is used in conjunction with the MQGMO_WAIT or MQGMO_SET_SIGNAL option. It is ignored if neither of these is specified. If one of these is specified, *WaitInterval* must be greater than or equal to zero, or the following special value:

MQWI_UNLIMITED

Unlimited wait interval.

The initial value of this field is 0.

Initial values and language declarations for MQGMO

Table 41. Initial values of fields in MQGMO for MQGMO

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQGMO_STRUC_ID	'GM0b'
<i>Version</i>	MQGMO_VERSION_1	1
<i>Options</i>	MQGMO_NO_WAIT	0
<i>WaitInterval</i>	None	0
<i>Signal1</i>	None	Null pointer on z/OS; 0 otherwise
<i>Signal2</i>	None	0
<i>ResolvedQName</i>	None	Null string or blanks
<i>MatchOptions</i>	MQMO_MATCH_MSG_ID + MQMO_MATCH_CORREL_ID	3
<i>GroupStatus</i>	MQGS_NOT_IN_GROUP	'b'
<i>SegmentStatus</i>	MQSS_NOT_A_SEGMENT	'b'
<i>Segmentation</i>	MQSEG_INHIBITED	'b'
<i>Reserved1</i>	None	'b'
<i>MsgToken</i>	MQMTOK_NONE	Nulls
<i>ReturnedLength</i>	MQRL_UNDEFINED	-1
<i>Reserved2</i>	None	'b'

Table 41. Initial values of fields in MQGMO for MQGMO (continued)

Field name	Name of constant	Value of constant
<i>MsgHandle</i>	MQHM_NONE	0
Notes: <ol style="list-style-type: none"> 1. The symbol <i>b</i> represents a single blank character. 2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages. 3. In the C programming language, the macro variable MQGMO_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure: MQGMO MyGMO = {MQGMO_DEFAULT}; 		

C declaration

```
typedef struct tagMQGMO MQGMO;
struct tagMQGMO {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQLONG    Options;          /* Options that control the action of
                                MQGET */
    MQLONG    WaitInterval;     /* Wait interval */
    MQLONG    Signal1;          /* Signal */
    MQLONG    Signal2;          /* Signal identifier */
    MQCHAR48  ResolvedQName;    /* Resolved name of destination queue */
    MQLONG    MatchOptions;     /* Options controlling selection criteria
                                used for MQGET */
    MQCHAR    GroupStatus;      /* Flag indicating whether message
                                retrieved is in a group */
    MQCHAR    SegmentStatus;    /* Flag indicating whether message
                                retrieved is a segment of a logical
                                message */
    MQCHAR    Segmentation;     /* Flag indicating whether further
                                segmentation is allowed for the message
                                retrieved */
    MQCHAR    Reserved1;        /* Reserved */
    MQBYTE16  MsgToken;         /* Message token */
    MQLONG    ReturnedLength;   /* Length of message data returned
                                (bytes) */
    MQLONG    Reserved2;        /* Reserved */
    MQHMSG    MsgHandle;        /* Message handle */
};
```

- On z/OS, the *Signal1* field is declared as PMQLONG.

COBOL declaration

```
** MQGMO structure
10 MQGMO.
** Structure identifier
15 MQGMO-STRUCID PIC X(4).
** Structure version number
15 MQGMO-VERSION PIC S9(9) BINARY.
** Options that control the action of MQGET
15 MQGMO-OPTIONS PIC S9(9) BINARY.
** Wait interval
15 MQGMO-WAITINTERVAL PIC S9(9) BINARY.
** Signal
15 MQGMO-SIGNAL1 PIC S9(9) BINARY.
** Signal identifier
15 MQGMO-SIGNAL2 PIC S9(9) BINARY.
** Resolved name of destination queue
```

```

15 MQGMO-RESOLVEDQNAME PIC X(48).
** Options controlling selection criteria used for MQGET
15 MQGMO-MATCHOPTIONS PIC S9(9) BINARY.
** Flag indicating whether message retrieved is in a group
15 MQGMO-GROUPSTATUS PIC X.
** Flag indicating whether message retrieved is a segment of a
** logical message
15 MQGMO-SEGMENTSTATUS PIC X.
** Flag indicating whether further segmentation is allowed for the
** message retrieved
15 MQGMO-SEGMENTATION PIC X.
** Reserved
15 MQGMO-RESERVED1 PIC X.
** Message token
15 MQGMO-MSGTOKEN PIC X(16).
** Length of message data returned (bytes)
15 MQGMO-RETURNEDLENGTH PIC S9(9) BINARY.
** Reserved
15 MQGMO-RESERVED2 PIC S9(9) BINARY.
** Message handle
15 MQGMO-MSGHANDLE PIC S9(18) BINARY.

```

- On z/OS, the *Signal1* field is declared as POINTER.

PL/I declaration

```

dcl
1 MQGMO based,
3 StrucId      char(4),      /* Structure identifier */
3 Version      fixed bin(31), /* Structure version number */
3 Options      fixed bin(31), /* Options that control the action of
                             MQGET */
3 WaitInterval fixed bin(31), /* Wait interval */
3 Signal1      fixed bin(31), /* Signal */
3 Signal2      fixed bin(31), /* Signal identifier */
3 ResolvedQName char(48),    /* Resolved name of destination
                             queue */
3 MatchOptions fixed bin(31), /* Options controlling selection
                             criteria used for MQGET */
3 GroupStatus  char(1),      /* Flag indicating whether message
                             retrieved is in a group */
3 SegmentStatus char(1),     /* Flag indicating whether message
                             retrieved is a segment of a logical
                             message */
3 Segmentation char(1),     /* Flag indicating whether further
                             segmentation is allowed for the
                             message retrieved */
3 Reserved1    char(1),      /* Reserved */
3 MsgToken     char(16),     /* Message token */
3 ReturnedLength fixed bin(31); /* Length of message data returned
                             (bytes) */
3 Reserved2    fixed bin(31); /* Reserved */
3 MsgHandle    fixed bin(63); /* Message handle */

```

- On z/OS, the *Signal1* field is declared as pointer.

System/390 assembler declaration

```

MQGMO          DSECT
MQGMO_STRUCID  DS CL4  Structure identifier
MQGMO_VERSION  DS F    Structure version number
MQGMO_OPTIONS  DS F    Options that control the action of
*              MQGET
MQGMO_WAITINTERVAL DS F  Wait interval
MQGMO_SIGNAL1  DS F    Signal
MQGMO_SIGNAL2  DS F    Signal identifier
MQGMO_RESOLVEDQNAME DS CL48 Resolved name of destination queue
MQGMO_MATCHOPTIONS DS F  Options controlling selection criteria
*              used for MQGET

```

```

MQGMO_GROUPSTATUS    DS   CL1  Flag indicating whether message
*                    retrieved is in a group
MQGMO_SEGMENTSTATUS  DS   CL1  Flag indicating whether message
*                    retrieved is a segment of a logical
*                    message
MQGMO_SEGMENTATION   DS   CL1  Flag indicating whether further
*                    segmentation is allowed for the message
*                    retrieved
MQGMO_RESERVED1      DS   CL1  Reserved
MQGMO_MSGTOKEN        DS  XL16  Message token
MQGMO_RETURNEDLENGTH DS   F    Length of message data returned (bytes)
MQGMO_RESERVED2      DS   F    Reserved
MQGMO_MSGHANDLE       DS   D    Message handle
MQGMO_LENGTH          EQU  *-MQGMO
                     ORG  MQGMO
MQGMO_AREA            DS   CL(MQGMO_LENGTH)

```

Visual Basic declaration

```

Type MQGMO
  StrucId      As String*4  'Structure identifier'
  Version      As Long      'Structure version number'
  Options      As Long      'Options that control the action of MQGET'
  WaitInterval As Long      'Wait interval'
  Signal1      As Long      'Signal'
  Signal2      As Long      'Signal identifier'
  ResolvedQName As String*48 'Resolved name of destination queue'
  MatchOptions As Long      'Options controlling selection criteria'
  'used for MQGET'
  GroupStatus  As String*1  'Flag indicating whether message'
  'retrieved is in a group'
  SegmentStatus As String*1 'Flag indicating whether message'
  'retrieved is a segment of a logical'
  'message'
  Segmentation As String*1  'Flag indicating whether further'
  'segmentation is allowed for the message'
  'retrieved'
  Reserved1    As String*1  'Reserved'
  MsgToken     As MQBYTE16 'Message token'
  ReturnedLength As Long    'Length of message data returned (bytes)'
End Type

```

MQIIH – IMS information header

The following table summarizes the fields in the structure.

Table 42. Fields in MQIIH

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>StrucLength</i>	Length of MQIIH structure	StrucLength
<i>Encoding</i>	Reserved	Encoding
<i>CodedCharSetId</i>	Reserved	CodedCharSetId
<i>Format</i>	MQ format name of data that follows MQIIH	Format
<i>Flags</i>	Flags	Flags
<i>LTermOverride</i>	Logical terminal override	LTermOverride
<i>MFSMapName</i>	Message format services map name	MFSMapName
<i>ReplyToFormat</i>	MQ format name of reply message	ReplyToFormat

Table 42. Fields in MQIIH (continued)

Field	Description	Topic
<i>Authenticator</i>	RACF™ password or passticket	Authenticator
<i>TranInstanceId</i>	Transaction instance identifier	TranInstanceId
<i>TranState</i>	Transaction state	TranState
<i>CommitMode</i>	Commit mode	CommitMode
<i>SecurityScope</i>	Security scope	SecurityScope
<i>Reserved</i>	Reserved	Reserved

Overview for MQIIH

Availability: All WebSphere MQ systems and WebSphere MQ clients.

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

Format name: MQFMT_IMS.

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

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

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

- The application message data following the MQIIH structure must be in the same character set and encoding as the MQIIH structure. Do not use the *CodedCharSetId* and *Encoding* fields in the MQIIH structure to specify the character set and encoding of the application message data.

You must provide a data-conversion exit to convert the application message data if the data is not one of the built-in formats supported by the queue manager.

Fields for MQIIH

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

Authenticator (MQCHAR8)

This is the RACF® password or passticket. It 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 can be used:

MQIAUT_NONE

No authentication.

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

The length of this field is given by `MQ_AUTHENTICATOR_LENGTH`. The initial value of this field is `MQIAUT_NONE`.

CodedCharSetId (MQLONG)

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

The Character Set Id for supported structures which follow a MQIIH structure is the same as that of the MQIIH structure itself and taken from any preceding MQ header.

CommitMode (MQCHAR)

This is the IMS commit mode. See the *OTMA Reference* for more information about IMS commit modes. The value must be one of the following:

MQICM_COMMIT_THEN_SEND

Commit then send.

This mode implies double queuing of output, but shorter region occupancy times. Fast-path and conversational transactions cannot run with this mode.

MQICM_SEND_THEN_COMMIT

Send then commit.

Any IMS transaction initiated as a result of a commit mode of `MQICM_SEND_THEN_COMMIT` runs in `RESPONSE` mode regardless of how the transaction is defined in the IMS system definition (`MSGTYPE` parameter in the `TRANSACT` macro). This also applies to transactions initiated by means of a transaction switch.

The initial value of this field is `MQICM_COMMIT_THEN_SEND`.

Encoding (MQLONG)

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

The Encoding for supported structures which follow a MQIIH structure is the same as that of the MQIIH structure itself and taken from any preceding MQ header.

Flags (MQLONG)

The flags value must be:

MQIIH_NONE

No flags.

MQIIH_PASS_EXPIRATION

The reply message contains:

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

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

MQIIH_REPLY_FORMAT_NONE

Sets the MQIIH.Format field of the reply to MQFMT_NONE.

MQIIH_IGNORE_PURG

Sets the TMAMIPRG indicator in the OTMA prefix, which requests that OTMA ignores PURG calls on the TP PCB for CM0 transactions.

The initial value of this field is MQIIH_NONE.

Format (MQCHAR8)

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

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

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

LTermOverride (MQCHAR8)

The logical terminal override, placed in the IO PCB field. It is optional; if it is not specified, the TPIPE name is used. It is ignored if the first byte is blank, or null.

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

MFSMapName (MQCHAR8)

The message format services map name, placed in the IO PCB field. It is optional. On input it represents the MID, on output it represents the MOD. It is ignored if the first byte is blank or null.

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

ReplyToFormat (MQCHAR8)

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

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

Reserved (MQCHAR)

This is a reserved field; it must be blank.

SecurityScope (MQCHAR)

This indicates the IMS security processing required. The following values are defined:

MQISS_CHECK

Check security scope: an ACEE is built in the control region, but not in the dependent region.

MQISS_FULL

Full security scope: a cached ACEE is built in the control region and a non-cached ACEE is built in the dependent region. If you use MQISS_FULL, ensure that the user ID for which the ACEE is built has access to the resources used in the dependent region.

If neither MQISS_CHECK nor MQISS_FULL is specified for this field, MQISS_CHECK is assumed.

The initial value of this field is MQISS_CHECK.

StrucId (MQCHAR4)

This is the structure identifier. The value must be:

MQIIH_STRUC_ID

Identifier for the IMS information header structure.

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

The initial value of this field is MQIIH_STRUC_ID.

StrucLength (MQLONG)

This is the length of MQIIH structure. The value must be:

MQIIH_LENGTH_1

Length of the IMS information header structure.

The initial value of this field is MQIIH_LENGTH_1.

TranInstanceld (MQBYTE16)

This is the transaction instance identifier. This field is used by output messages from IMS, so is ignored on first input. If you set *TranState* to MQITS_IN_CONVERSATION, this must be provided in the next input, and all subsequent inputs, to enable IMS to correlate the messages to the correct conversation. You can use the following special value:

MQITII_NONE

No transaction instance identifier.

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

The length of this field is given by MQ_TRAN_INSTANCE_ID_LENGTH. The initial value of this field is MQITII_NONE.

TranState (MQCHAR)

This indicates the IMS conversation state. This is ignored on first input because no conversation exists. On subsequent inputs it indicates whether a conversation is active or not. On output it is set by IMS. The value must be one of the following:

MQITS_IN_CONVERSATION

In conversation.

MQITS_NOT_IN_CONVERSATION

Not in conversation.

MQITS_ARCHITECTED

Return transaction state data in architected form.

This value is used only with the IMS /DISPLAY TRAN command. It returns the transaction state data in the IMS architected form instead of character form. See the *WebSphere MQ Application Programming Guide* for further details.

The initial value of this field is MQITS_NOT_IN_CONVERSATION.

Version (MQLONG)

This is the structure version number. The value must be:

MQIIH_VERSION_1

Version number for IMS information header structure.

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

MQIIH_CURRENT_VERSION

Current version of IMS information header structure.

The initial value of this field is MQIIH_VERSION_1.

Initial values and language declarations for MQIIH

Table 43. Initial values of fields in MQIIH for MQIIH

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQIIH_STRUC_ID	'IIHb'
<i>Version</i>	MQIIH_VERSION_1	1
<i>StrucLength</i>	MQIIH_LENGTH_1	84
<i>Encoding</i>	None	0
<i>CodedCharSetId</i>	None	0
<i>Format</i>	MQFMT_NONE	Blanks
<i>Flags</i>	MQIIH_NONE	0
<i>LTermOverride</i>	None	Blanks
<i>MFSMapName</i>	None	Blanks
<i>ReplyToFormat</i>	MQFMT_NONE	Blanks
<i>Authenticator</i>	MQIAUT_NONE	Blanks
<i>TranInstanceId</i>	MQITII_NONE	Nulls
<i>TranState</i>	MQITS_NOT_IN_CONVERSATION	'b'
<i>CommitMode</i>	MQICM_COMMIT_THEN_SEND	'0'

Table 43. Initial values of fields in MQIIH for MQIIH (continued)

Field name	Name of constant	Value of constant
<i>SecurityScope</i>	MQISS_CHECK	'C'
<i>Reserved</i>	None	'b'

Notes:

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

```
MQIIH MyIIH = {MQIIH_DEFAULT};
```

C declaration

```
typedef struct tagMQIIH MQIIH;
struct tagMQIIH {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQLONG    StrucLength;      /* Length of MQIIH structure */
    MQLONG    Encoding;         /* Reserved */
    MQLONG    CodedCharSetId;   /* Reserved */
    MQCHAR8   Format;           /* MQ format name of data that follows
                               MQIIH */
    MQLONG    Flags;            /* Flags */
    MQCHAR8   LTermOverride;    /* Logical terminal override */
    MQCHAR8   MFSMapName;       /* Message format services map name */
    MQCHAR8   ReplyToFormat;    /* MQ format name of reply message */
    MQCHAR8   Authenticator;     /* RACF password or passticket */
    MQBYTE16  TranInstanceId;    /* Transaction instance identifier */
    MQCHAR    TranState;        /* Transaction state */
    MQCHAR    CommitMode;       /* Commit mode */
    MQCHAR    SecurityScope;    /* Security scope */
    MQCHAR    Reserved;         /* Reserved */
};
```

COBOL declaration

```
** MQIIH structure
10 MQIIH.
** Structure identifier
15 MQIIH-STRUCID PIC X(4).
** Structure version number
15 MQIIH-VERSION PIC S9(9) BINARY.
** Length of MQIIH structure
15 MQIIH-STRUCLength PIC S9(9) BINARY.
** Reserved
15 MQIIH-ENCODING PIC S9(9) BINARY.
** Reserved
15 MQIIH-CODEDCHARSETID PIC S9(9) BINARY.
** MQ format name of data that follows MQIIH
15 MQIIH-FORMAT PIC X(8).
** Flags
15 MQIIH-FLAGS PIC S9(9) BINARY.
** Logical terminal override
15 MQIIH-LTERMOVERRIDE PIC X(8).
** Message format services map name
15 MQIIH-MFSMAPNAME PIC X(8).
** MQ format name of reply message
15 MQIIH-REPLYTOFORMAT PIC X(8).
** RACF password or passticket
15 MQIIH-AUTHENTICATOR PIC X(8).
** Transaction instance identifier
```

```

15 MQIIH-TRANINSTANCEID PIC X(16).
** Transaction state
15 MQIIH-TRANSTATE PIC X.
** Commit mode
15 MQIIH-COMMITMODE PIC X.
** Security scope
15 MQIIH-SECURITYSCOPE PIC X.
** Reserved
15 MQIIH-RESERVED PIC X.

```

PL/I declaration

```

dc1
1 MQIIH based,
3 StrucId char(4), /* Structure identifier */
3 Version fixed bin(31), /* Structure version number */
3 StrucLength fixed bin(31), /* Length of MQIIH structure */
3 Encoding fixed bin(31), /* Reserved */
3 CodedCharSetId fixed bin(31), /* Reserved */
3 Format char(8), /* MQ format name of data that follows
MQIIH */
3 Flags fixed bin(31), /* Flags */
3 LTermOverride char(8), /* Logical terminal override */
3 MFMapName char(8), /* Message format services map name */
3 ReplyToFormat char(8), /* MQ format name of reply message */
3 Authenticator char(8), /* RACF password or passticket */
3 TranInstanceId char(16), /* Transaction instance identifier */
3 TranState char(1), /* Transaction state */
3 CommitMode char(1), /* Commit mode */
3 SecurityScope char(1), /* Security scope */
3 Reserved char(1); /* Reserved */

```

System/390 assembler declaration

```

MQIIH DSECT
MQIIH_STRUCID DS CL4 Structure identifier
MQIIH_VERSION DS F Structure version number
MQIIH_STRUCLNGTH DS F Length of MQIIH structure
MQIIH_ENCODING DS F Reserved
MQIIH_CODEDCHARSETID DS F Reserved
MQIIH_FORMAT DS CL8 MQ format name of data that follows
* MQIIH
MQIIH_FLAGS DS F Flags
MQIIH_LTERM_OVERRIDE DS CL8 Logical terminal override
MQIIH_MFMAPNAME DS CL8 Message format services map name
MQIIH_REPLYTOFORMAT DS CL8 MQ format name of reply message
MQIIH_AUTHENTICATOR DS CL8 RACF password or passticket
MQIIH_TRANINSTANCEID DS XL16 Transaction instance identifier
MQIIH_TRANSTATE DS CL1 Transaction state
MQIIH_COMMITMODE DS CL1 Commit mode
MQIIH_SECURITYSCOPE DS CL1 Security scope
MQIIH_RESERVED DS CL1 Reserved
*
MQIIH_LENGTH EQU *-MQIIH
ORG MQIIH
MQIIH_AREA DS CL(MQIIH_LENGTH)

```

Visual Basic declaration

```

Type MQIIH
StrucId As String*4 'Structure identifier'
Version As Long 'Structure version number'
StrucLength As Long 'Length of MQIIH structure'
Encoding As Long 'Reserved'
CodedCharSetId As Long 'Reserved'
Format As String*8 'MQ format name of data that follows MQIIH'
Flags As Long 'Flags'
LTermOverride As String*8 'Logical terminal override'
MFMapName As String*8 'Message format services map name'

```



```

ReplyToFormat As String*8 'MQ format name of reply message'
Authenticator As String*8 'RACF password or passticket'
TranInstanceId As MQBYTE16 'Transaction instance identifier'
TranState As String*1 'Transaction state'
CommitMode As String*1 'Commit mode'
SecurityScope As String*1 'Security scope'
Reserved As String*1 'Reserved'
End Type

```

MQIMPO – Inquire message property options

The following table summarizes the fields in the structure. MQIMPO structure - inquire message property options

Table 44. Fields in MQIMPO

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>Options</i>	Options controlling the action of MQINQMP	Options
<i>RequestedEncoding</i>	Encoding into which the enquired property is to be converted	RequestedEncoding
<i>RequestedCCSID</i>	Character set of the inquired property	RequestedCCSID
<i>ReturnedEncoding</i>	Encoding of the returned value	ReturnedEncoding
<i>ReturnedCCSID</i>	Character set of returned value	ReturnedCCSID
<i>Reserved1</i>	Reserved field	ReturnedCCSID
<i>ReturnedName</i>	Name of the inquired property	ReturnedName
<i>TypeString</i>	String representation of the data type of the property	TypeString

Overview for MQIMPO

The inquire message properties options structure.

Availability: All WebSphere MQ systems and WebSphere MQ clients.

Purpose: The MQIMPO structure allows applications to specify options that control how properties of messages are inquired. The structure is an input parameter on the MQINQMP call.

Character set and encoding: Data in MQIMPO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields for MQIMPO

Inquire message property options structure - fields

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

Options (MQLONG)

Inquire message property options structure - Options field

The following options control the action of MQINQMP. You can specify one or more of these options, and if you need more than one, the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

Combinations of options that are not valid are noted; all other combinations are valid.

Value data options: The following options relate to the processing of the value data when the property is retrieved from the message.

MQIMPO_CONVERT_VALUE

This option requests that the value of the property be converted to conform to the *RequestedCCSID* and *RequestedEncoding* values specified before the MQINQMP call returns the property value in the *Value* area.

- If conversion is successful, the *ReturnedCCSID* and *ReturnedEncoding* fields are set to the same as *RequestedCCSID* and *RequestedEncoding* on return from the MQINQMP call.
- If conversion fails, but the MQINQMP call otherwise completes without error, the property value is returned unconverted.

If the property is a string, the *ReturnedCCSID* and *ReturnedEncoding* fields are set to the character set and encoding of the unconverted string. The completion code is MQCC_WARNING in this case, with reason code MQRC_PROP_VALUE_NOT_CONVERTED. The property cursor is advanced to the returned property.

If the property value expands during conversion, and exceeds the size of the *Value* parameter, the value is returned unconverted, with completion code MQCC_FAILED; the reason code is set to MQRC_PROPERTY_VALUE_TOO_BIG.

The *DataLength* parameter of the MQINQMP call returns the length that the property value would have converted to, in order to allow the application to determine the size of the buffer required to accommodate the converted property value. The property cursor is unchanged.

This option also requests that:

- If the property name contains a wildcard, and
- The *ReturnedName* field is initialized with an address or offset for the returned name,

then the returned name is converted to conform to the *RequestedCCSID* and *RequestedEncoding* values.

- If conversion is successful, the *VSCSID* field of *ReturnedName* and the encoding of the returned name are set to the input value of *RequestedCCSID* and *RequestedEncoding*.
- If conversion fails, but the MQINQMP call otherwise completes without error or warning, the returned name is unconverted. The completion code is MQCC_WARNING in this case, with reason code MQRC_PROP_NAME_NOT_CONVERTED.

The property cursor is advanced to the returned property. MQRC_PROP_VALUE_NOT_CONVERTED is returned if both the value and the name are not converted.

If the returned name expands during conversion, and exceeds the size of the *VSBufsize* field of the *RequestedName*, the returned string is left unconverted, with completion code MQCC_FAILED and the reason code is set to MQRC_PROPERTY_NAME_TOO_BIG.

The *VSLength* field of the MQCHARV structure returns the length that the property value would have converted to, in order to allow the application to determine the size of the buffer required to accommodate the converted property value. The property cursor is unchanged.

MQIMPO_CONVERT_TYPE

This option requests that the value of the property be converted from its current data type, into the data type specified on the *Type* parameter of the MQINQMP call.

- If conversion is successful, the *Type* parameter is unchanged on return of the MQINQMP call.
- If conversion fails, but the MQINQMP call otherwise completes without error, the call fails with reason MQRC_PROP_CONV_NOT_SUPPORTED. The property cursor is unchanged.

If the conversion of the data type causes the value to expand during conversion, and the converted value exceeds the size of the *Value* parameter, the value is returned unconverted, with completion code MQCC_FAILED and the reason code is set to MQRC_PROPERTY_VALUE_TOO_BIG.

The *DataLength* parameter of the MQINQMP call returns the length that the property value would have converted to, in order to allow the application to determine the size of the buffer required to accommodate the converted property value. The property cursor is unchanged.

If the value of the *Type* parameter of the MQINQMP call is not valid, the call fails with reason MQRC_PROPERTY_TYPE_ERROR.

If the requested data type conversion is not supported, the call fails with reason MQRC_PROP_CONV_NOT_SUPPORTED. The following data type conversions are supported:

Property data type	Supported target data types
MQTYPE_BOOLEAN	MQTYPE_STRING, MQTYPE_INT8, MQTYPE_INT16, MQTYPE_INT32, MQTYPE_INT64
MQTYPE_BYTE_STRING	MQTYPE_STRING
MQTYPE_INT8	MQTYPE_STRING, MQTYPE_INT16, MQTYPE_INT32, MQTYPE_INT64
MQTYPE_INT16	MQTYPE_STRING, MQTYPE_INT32, MQTYPE_INT64
MQTYPE_INT32	MQTYPE_STRING, MQTYPE_INT64
MQTYPE_INT64	MQTYPE_STRING
MQTYPE_FLOAT32	MQTYPE_STRING, MQTYPE_FLOAT64
MQTYPE_FLOAT64	MQTYPE_STRING
MQTYPE_STRING	MQTYPE_BOOLEAN, MQTYPE_INT8, MQTYPE_INT16, MQTYPE_INT32, MQTYPE_INT64, MQTYPE_FLOAT32, MQTYPE_FLOAT64
MQTYPE_NULL	None

The general rules governing the supported conversions are as follows:

- Numeric property values can be converted from one data type to another, provided that no data is lost during the conversion.
For example, the value of a property with data type MQTYPE_INT32 can be converted into a value with data type MQTYPE_INT64, but cannot be converted into a value with data type MQTYPE_INT16.
- A property value of any data type can be converted into a string.
- A string property value can be converted to any other data type provided the string is formatted correctly for the conversion. If an application attempts to convert a string property value that is not formatted correctly, WebSphere MQ returns reason code MQRC_PROP_NUMBER_FORMAT_ERROR.
- If an application attempts a conversion that is not supported, WebSphere MQ returns reason code MQRC_PROP_CONV_NOT_SUPPORTED.

The specific rules for converting a property value from one data type to another are as follows:

- When converting an MQTYPE_BOOLEAN property value to a string, the value TRUE is converted to the string "TRUE", and the value false is converted to the string "FALSE".
- When converting an MQTYPE_BOOLEAN property value to a numeric data type, the value TRUE is converted to one, and the value FALSE is converted to zero.
- When converting a string property value to an MQTYPE_BOOLEAN value, the string "TRUE" , or "1" , is converted to TRUE, and the string "FALSE", or "0", is converted to FALSE.

Note that the terms "TRUE" and "FALSE" are not case sensitive.

Any other string cannot be converted; WebSphere MQ returns reason code MQRC_PROP_NUMBER_FORMAT_ERROR.

- When converting a string property value to a value with data type MQTYPE_INT8, MQTYPE_INT16, MQTYPE_INT32 or MQTYPE_INT64, the string must have the following format:

[blanks][sign]digits

The meanings of the components of the string are as follows:

blanks Optional leading blank characters

sign An optional plus sign (+) or minus sign (-) character.

digits A contiguous sequence of digit characters (0-9). At least one digit character must be present.

After the sequence of digit characters, the string can contain other characters that are not digit characters, but the conversion stops as soon as the first of these characters is reached. The string is assumed to represent a decimal integer.

WebSphere MQ returns reason code

MQRC_PROP_NUMBER_FORMAT_ERROR if the string is not formatted correctly.

- When converting a string property value to a value with data type MQTYPE_FLOAT32 or MQTYPE_FLOAT64, the string must have the following format:

[blanks][sign]digits[.digits][e_char[e_sign]e_digits]

The meanings of the components of the string are as follows:

blanks Optional leading blank characters

sign An optional plus sign (+) or minus sign (-) character.

digits A contiguous sequence of digit characters (0-9). At least one digit character must be present.

e_char An exponent character, which is either "E" or "e".

e_sign An optional plus sign (+) or minus sign (-) character for the exponent.

e_digits

A contiguous sequence of digit characters (0-9) for the exponent. At least one digit character must be present if the string contains an exponent character.

After the sequence of digit characters, or the optional characters representing an exponent, the string can contain other characters that are not digit characters, but the conversion stops as soon as the first of these characters is reached. The string is assumed to represent a decimal floating point number with an exponent that is a power of 10.

WebSphere MQ returns reason code

MQRC_PROP_NUMBER_FORMAT_ERROR if the string is not formatted correctly.

- When converting a numeric property value to a string, the value is converted to the string representation of the value as a decimal number, not the string containing the ASCII character for that value. For example, the integer 65 is converted to the string "65", not the string "A".
- When converting a byte string property value to a string, each byte is converted to the two hexadecimal characters that represent the byte. For example, the byte array {0xF1, 0x12, 0x00, 0xFF} is converted to the string "F11200FF".

MQIMPO_QUERY_LENGTH

Query the type and length of the property value. The length is returned in the *DataLength* parameter of the MQINQMP call. The property value is not returned.

If a *ReturnedName* buffer is specified, the *VSLength* field of the MQCHARV structure is filled in with the length of the property name. The property name is not returned.

Iteration options: The following options relate to iterating over properties, using a name with a wildcard character

MQIMPO_INQ_FIRST

Inquire on the first property that matches the specified name. After this call, a cursor is established on the property that is returned.

This is the default value.

The MQIMPO_INQ_PROP_UNDER_CURSOR option can subsequently be used with an MQINQMP call, if required, to inquire on the same property again.

Note that there is only one property cursor; therefore, if the property name, specified in the MQINQMP call, changes the cursor is reset.

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

MQIMPO_INQ_NEXT

MQIMPO_INQ_PROP_UNDER_CURSOR

MQIMPO_INQ_NEXT

Inquires on the next property that matches the specified name, continuing the search from the property cursor. The cursor is advanced to the property that is returned.

If this is the first MQINQMP call for the specified name, then the first property that matches the specified name is returned.

The MQIMPO_INQ_PROP_UNDER_CURSOR option can subsequently be used with an MQINQMP call if required, to inquire on the same property again.

If the property under the cursor has been deleted, MQINQMP returns the next matching property following the one that has been deleted.

If a property is added that matches the wildcard, while an iteration is in progress, the property might or might not be returned during the completion of the iteration. The property is returned once the iteration restarts using MQIMPO_INQ_FIRST.

A property matching the wildcard that was deleted, while the iteration was in progress, is not returned subsequent to its deletion.

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

MQIMPO_INQ_FIRST

MQIMPO_INQ_PROP_UNDER_CURSOR

MQIMPO_INQ_PROP_UNDER_CURSOR

Retrieve the value of the property pointed to by the property cursor. The property pointed to by the property cursor is the one that was last inquired, using either the MQIMPO_INQ_FIRST or the MQIMPO_INQ_NEXT option.

The property cursor is reset when the message handle is reused, when the message handle is specified in the *MsgHandle* field of the MQGMO on an MQGET call, or when the message handle is specified in *OriginalMsgHandle* or *NewMsgHandle* fields of the MQPMO structure on an MQPUT call.

If this option is used when the property cursor has not yet been established, or if the property pointed to by the property cursor has been deleted, the call fails with completion code MQCC_FAILED and reason MQRC_PROPERTY_NOT_AVAILABLE.

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

MQIMPO_INQ_FIRST

MQIMPO_INQ_NEXT

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

MQIMPO_NONE

Use this value to indicate that no other options have been specified; all options assume their default values.

MQIMPO_NONE aids 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 always an input field. The initial value of this field is MQIMPO_INQ_FIRST.

RequestedCCSID (MQLONG)

Inquire message property options structure - RequestedCCSID field

The character set that the inquired property value is to be converted into if the value is a character string. This is also the character set into which the *ReturnedName* is to be converted when MQIMPO_CONVERT_VALUE or MQIMPO_CONVERT_TYPE is specified.

The initial value of this field is MQCCSI_APPL.

RequestedEncoding (MQLONG)

Inquire message property options structure - RequestedEncoding field

This is the encoding into which the inquired property value is to be converted when MQIMPO_CONVERT_VALUE or MQIMPO_CONVERT_TYPE is specified.

The initial value of this field is MQENC_NATIVE.

Reserved1 (MQCHAR)

This is a reserved field. The initial value of this field is a blank character.

ReturnedCCSID (MQLONG)

Inquire message property options structure - ReturnedCCSID field

On output, this is the character set of the value returned if the *Type* parameter of the MQINQMP call is MQTYPE_STRING.

If the MQIMPO_CONVERT_VALUE option is specified and conversion was successful, the *ReturnedCCSID* field, on return, is the same value as the value passed in.

The initial value of this field is zero.

ReturnedEncoding (MQLONG)

Inquire message property options structure - ReturnedEncoding field

On output, this is the encoding of the value returned.

If the MQIMPO_CONVERT_VALUE option is specified and conversion was successful, the *ReturnedEncoding* field, on return, is the same value as the value passed in.

The initial value of this field is MQENC_NATIVE.

ReturnedName (MQCHARV)

Inquire message property options structure - ReturnedName field

The actual name of the inquired property.

On input a string buffer can be passed in using the *VSPtr* or *VSoffset* field of the MQCHARV structure. The length of the string buffer is specified using the *VSBufferSize* field of the MQCHARV structure.

On return from the MQINQMP call, the string buffer is completed with the name of the property that was inquired, provided the string buffer was long enough to fully contain the name. The *VSLength* field of the MQCHARV structure is filled in with the length of the property name. The *VSCSID* field of the MQCHARV structure is filled in to indicate the character set of the returned name, whether or not conversion of the name failed.

This is an input/output field. The initial value of this field is MQCHARV_DEFAULT.

StrucId (MQCHAR4)

Inquire message property options structure - StrucId field

This is the structure identifier. The value must be:

MQIMPO_STRUC_ID

Identifier for inquire message property options structure.

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

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

TypeString (MQCHAR8)

Inquire message property options structure - TypeString field

A string representation of the data type of the property.

If the property was specified in an MQRFH2 header and the MQRFH2 dt attribute is not recognized, this field can be used to determine the data type of the property. *TypeString* is returned in coded character set 1208 (UTF-8), and is the first eight bytes of the value of the dt attribute of the property that failed to be recognized

This is always an output field. The initial value of this field is the null string in the C programming language, and 8 blank characters in other programming languages.

Version (MQLONG)

Inquire message property options structure - Version field

This is the structure version number. The value must be:

MQIMPO_VERSION_1

Version number for inquire message property options structure.

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

MQIMPO_CURRENT_VERSION

Current version of inquire message property options structure.

This is always an input field. The initial value of this field is MQIMPO_VERSION_1.

Initial values and language declarations for MQIMPO

Inquire message property options structure - Initial values

Table 45. Initial values of fields in MQIMPO

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQIMPO_STRUC_ID	'IMPO'
<i>Version</i>	MQIMPO_VERSION_1	1
<i>Options</i>	MQIMPO_INQ_FIRST	
<i>RequestedEncoding</i>	MQENC_NATIVE	
<i>RequestedCCSID</i>	MQCCSI_APPL	
<i>ReturnedEncoding</i>	MQENC_NATIVE	
<i>ReturnedCCSID</i>	0	
<i>Reserved1</i>	0	
<i>ReturnedName</i>	MQCHARV_DEFAULT	
<i>TypeString</i>	Null string or blanks	
Notes: <ol style="list-style-type: none"> 1. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages. 2. In the C programming language, the macro variable MQIMPO_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure: <pre>MQIMPO MyIMPO = {MQIMPO_DEFAULT};</pre> 		

C declaration

Inquire message property options structure - C language declaration

```
typedef struct tagMQIMPO MQIMPO;
struct tagMQIMPO {
    MQCHAR4  StrucId;           /* Structure identifier */
    MQLONG   Version;          /* Structure version number */
    MQLONG   Options;          /* Options that control the action of
                               MQINQMP */
    MQLONG   RequestedEncoding; /* Requested encoding of Value */
    MQLONG   RequestedCCSID;    /* Requested character set identifier
                               of Value */
    MQLONG   ReturnedEncoding; /* Returned encoding of Value */
    MQLONG   ReturnedCCSID;    /* Returned character set identifier
                               of Value */
    MQCHAR   Reserved1         /* Reserved field */
    MQCHARV  ReturnedName;     /* Returned property name */
    MQCHAR8  TypeString;       /* Property data type as a string */
};
```

COBOL declaration

Inquire message property options structure - COBOL language declaration

```
** MQIMPO structure
   10 MQIMPO.
**   Structure identifier
      15 MQIMPO-STRUCID          PIC X(4).
**   Structure version number
      15 MQIMPO-VERSION          PIC S9(9) BINARY.
**   Options that control the action of MQINQMP
      15 MQIMPO-OPTIONS          PIC S9(9) BINARY.
**   Requested encoding of VALUE
      15 MQIMPO-REQUESTEDENCODING PIC S9(9) BINARY.
**   Requested character set identifier of VALUE
      15 MQIMPO-REQUESTEDCCSID   PIC S9(9) BINARY.
```

```

** Returned encoding of VALUE
15 MQIMPO-RETURNEDENCODING PIC S9(9) BINARY.
** Returned character set identifier of VALUE
15 MQIMPO-RETURNEDCCSID PIC S9(9) BINARY.
** Reserved field
15 MQIMPO-RESERVED1
** Returned property name
15 MQIMPO-RETURNEDNAME.
** Address of variable length string
20 MQIMPO-RETURNEDNAME-VSPTR POINTER.
** Offset of variable length string
20 MQIMPO-RETURNEDNAME-VSOFFSET PIC S9(9) BINARY.
** CCSID of variable length string
20 MQIMPO-RETURNEDNAME-VSCCSID PIC S9(9) BINARY.
** Property data type as string
15 MQIMPO-TYPESTRING PIC S9(9) BINARY.

```

PL/I declaration

Inquire message property options structure - PL/I language declaration

```

dcl
1 MQIMPO based,
3 StrucId char(4), /* Structure identifier */
3 Version fixed bin(31), /* Structure version number */
3 Options fixed bin(31), /* Options that control the
action of MQINQMP */
3 RequestedEncoding fixed bin(31), /* Requested encoding of
Value */
3 RequestedCCSID fixed bin(31), /* Requested character set
identifier of Value */
3 ReturnedEncoding fixed bin(31), /* Returned encoding of
Value */
3 ReturnedCCSID fixed bin(31), /* Returned character set
identifier of Value */
3 Reserved1 fixed bin(31), /* Reserved field */
3 ReturnedName, /* Returned property name */
5 ReturnedName_VSPtr pointer, /* Address of returned
name */
5 5 ReturnedName_VSOffset fixed bin(31), /* Offset of returned
name */
5 5 ReturnedName_VSCCSID fixed bin(31), /* CCSID of returned
name */
3 TypeString char(8); /* Property data type as
string */

```

System/390 assembler declaration

Inquire message property options structure - Assembler language declaration

```

MQIMPO DSECT
MQIMPO_STRUCID DS CL4 Structure identifier
MQIMPO_VERSION DS F Structure version number
MQIMPO_OPTIONS DS F Options that control the
* action of MQINQMP
MQIMPO_REQUESTEDENCODING DS F Requested encoding of VALUE
MQIMPO_REQUESTEDCCSID DS F Requested character set
* identifier of VALUE
MQIMPO_RETURNEDENCODING DS F Returned encoding of VALUE
MQIMPO_RETURNEDCCSID DS F Returned character set
* identifier of VALUE
MQIMPO_RESERVED1 DS F Reserved field
MQIMPO_RETURNEDNAME DS 0F Force fullword alignment
MQIMPO_RETURNEDNAME_VSPTR DS F Address of returned name
MQIMPO_RETURNEDNAME_VSOFFSET DS F Offset of returned name
MQIMPO_RETURNEDNAME_VSLENGTH DS F Length of returned name
MQIMPO_RETURNEDNAME_VSCCSID DS F CCSID of returned name
MQIMPO_RETURNEDNAME_LENGTH EQU *-MQIMPO_RETURNEDNAME
ORG MQIMPO_RETURNEDNAME

```

MQIMPO_RETURNEDNAME_AREA	DS	CL(MQIMPO_RETURNEDNAME_LENGTH)
*		
MQIMPO_TYPESTRING	DS	CL8 Property data type as string
MQIMPO_LENGTH	EQU	*-MQIMPO
MQIMPO_AREA	DS	CL(MQIMPO_LENGTH)

MQMD – Message descriptor

The following table summarizes the fields in the structure.

Table 46. Fields in MQMD

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>Report</i>	Options for report messages	Report
<i>MsgType</i>	Message type	MsgType
<i>Expiry</i>	Message lifetime	MQMD - Expiry field
<i>Feedback</i>	Feedback or reason code	MQMD - Feedback field
<i>Encoding</i>	Numeric encoding of message data	Encoding
<i>CodedCharSetId</i>	Character set identifier of message data	CodedCharSetId
<i>Format</i>	Format name of message data	Format
<i>Priority</i>	Message priority	Priority
<i>Persistence</i>	Message persistence	Persistence
<i>MsgId</i>	Message identifier	MQMD - MsgId field
<i>CorrelId</i>	Correlation identifier	CorrelId
<i>BackoutCount</i>	Backout counter	BackoutCount
<i>ReplyToQ</i>	Name of reply queue	ReplyToQ
<i>ReplyToQMgr</i>	Name of reply queue manager	ReplyToQMgr
<i>UserIdentifier</i>	User identifier	UserIdentifier
<i>AccountingToken</i>	Accounting token	AccountingToken
<i>ApplIdentityData</i>	Application data relating to identity	ApplIdentityData
<i>PutApplType</i>	Type of application that put the message	PutApplType
<i>PutApplName</i>	Name of application that put the message	PutApplName
<i>PutDate</i>	Date when message was put	PutDate
<i>PutTime</i>	Time when message was put	PutTime
<i>ApplOriginData</i>	Application data relating to origin	ApplOriginData
Note: The remaining fields are ignored if <i>Version</i> is less than MQMD_VERSION_2.		
<i>GroupId</i>	Group identifier	GroupId
<i>MsgSeqNumber</i>	Sequence number of logical message within group	MsgSeqNumber
<i>Offset</i>	Offset of data in physical message from start of logical message	Offset
<i>MsgFlags</i>	Message flags	MQMD - MsgFlags field

Table 46. Fields in MQMD (continued)

Field	Description	Topic
<i>OriginalLength</i>	Length of original message	OriginalLength

Overview for MQMD

Availability: All WebSphere MQ systems, plus WebSphere MQ clients connected to these systems.

Purpose: The MQMD structure contains the control information that accompanies the application data when a message travels between the sending and receiving applications. The structure is an input/output parameter on the MQGET, MQPUT and MQPUT1 calls.

Version: The current version of MQMD is MQMD_VERSION_2. Applications that are intended to be portable between several environments must ensure that the required version of MQMD is supported in all of the environments concerned. Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions that follow.

The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQMD that is supported by the environment, but with the initial value of the *Version* field set to MQMD_VERSION_1. To use fields that are not present in the version-1 structure, the application must set the *Version* field to the version number of the version required.

A declaration for the version-1 structure is available with the name MQMD1.

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

If the sending and receiving queue managers use different character sets or encodings, the data in MQMD is converted automatically. It is not necessary for the application to convert the MQMD.

Using different versions of MQMD: A version-2 MQMD is generally equivalent to using a version-1 MQMD and prefixing the message data with an MQMDE structure. However, if all the fields in the MQMDE structure have their default values, the MQMDE can be omitted. A version-1 MQMD plus MQMDE are used as described below.

- On the MQPUT and MQPUT1 calls, if the application provides a version-1 MQMD, the application can optionally prefix the message data with an MQMDE, setting the *Format* field in MQMD to MQFMT_MD_EXTENSION to indicate that an MQMDE is present. If the application does not provide an MQMDE, the queue manager assumes default values for the fields in the MQMDE.

Note: Several of the fields that exist in the version-2 MQMD but not the version-1 MQMD are input/output fields on the MQPUT and MQPUT1 calls. However, the queue manager does *not* return any values in the equivalent fields

in the MQMDE on output from the MQPUT and MQPUT1 calls; if the application requires those output values, it must use a version-2 MQMD.

- On the MQGET call, if the application provides a version-1 MQMD, the queue manager prefixes the message returned with an MQMDE, but only if one or more of the fields in the MQMDE has a non-default value. The *Format* field in MQMD will have the value MQFMT_MD_EXTENSION to indicate that an MQMDE is present.

The default values that the queue manager uses for the fields in the MQMDE are the same as the initial values of those fields, shown in Table 50 on page 244.

When a message is on a transmission queue, some of the fields in MQMD are set to particular values; see “MQXQH – Transmission-queue header” on page 387 for details.

Message context: Certain fields in MQMD contain the message context. There are two types of message context: *identity context* and *origin context*. Usually:

- Identity context relates to the application that *originally* put the message
- Origin context relates to the application that *most recently* put the message.

These two applications can be the same application, but they can also be different applications (for example, when a message is forwarded from one application to another).

Although identity and origin context usually have the meanings described above, the content of both types of context fields in MQMD depends on the MQPMO_*_CONTEXT options that are specified when the message is put. As a result, identity context does not necessarily relate to the application that originally put the message, and origin context does not necessarily relate to the application that most-recently put the message; it depends on the design of the application suite.

The message channel agent (MCA) never alters message context. MCAs that receive messages from remote queue managers use the context option MQPMO_SET_ALL_CONTEXT on the MQPUT or MQPUT1 call. This allows the receiving MCA to preserve exactly the message context that travelled with the message from the sending MCA. However, the result is that the origin context does not relate to the application that most recently put the message (the receiving MCA), but instead relates to an earlier application that put the message (possibly the originating application itself).

In the descriptions below, the context fields are described as though they are used as described above. For more information about message context, see the *WebSphere MQ Application Programming Guide*.

Fields for MQMD

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

AccountingToken (MQBYTE32)

This is the accounting token, part of the **identity context** of the message. For more information about message context, see “Overview for MQMD” on page 182; also see the *WebSphere MQ Application Programming Guide*.

AccountingToken allows an application to charge appropriately for work done as a result of the message. The queue manager treats this information as a string of bits and does not check its content.

The queue manager generates this information as follows:

- The first byte of the field is set to the length of the accounting information present in the bytes that follow; this length is in the range zero through 30, and is stored in the first byte as a binary integer.
- The second and subsequent bytes (as specified by the length field) are set to the accounting information appropriate to the environment.
 - On z/OS the accounting information is set to:
 - For z/OS batch, the accounting information from the JES JOB card or from a JES ACCT statement in the EXEC card (comma separators are changed to X'FF'). This information is truncated, if necessary, to 31 bytes.
 - For TSO, the user's account number.
 - For CICS, the LU 6.2 unit of work identifier (UEPUOWDS) (26 bytes).
 - For IMS, the 8-character PSB name concatenated with the 16-character IMS recovery token.
 - On i5/OS, the accounting information is set to the accounting code for the job.
 - On UNIX systems, the accounting information is set to the numeric user identifier, in ASCII characters.
 - On Windows, the accounting information is set to a Windows security identifier (SID) in a compressed format. The SID uniquely identifies the user identifier stored in the *UserIdentifier* field. When the SID is stored in the *AccountingToken* field, the 6-byte Identifier Authority (located in the third and subsequent bytes of the SID) is omitted. For example, if the Windows SID is 28 bytes long, 22 bytes of SID information are stored in the *AccountingToken* field.
- The last byte (byte 32) of the accounting field is set to the accounting token type (in this case MQACTT_NT_SECURITY_ID, x '0b'):

MQACTT_CICS_LUOW_ID
CICS LUOW identifier.

MQACTT_NT_SECURITY_ID
Windows security identifier.

MQACTT_OS400_ACCOUNT_TOKEN
i5/OS accounting token.

MQACTT_UNIX_NUMERIC_ID
UNIX systems numeric identifier.

MQACTT_USER
User-defined accounting token.

MQACTT_UNKNOWN
Unknown accounting-token type.

The accounting-token type is set to an explicit value only in the following environments: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems. In other environments, the accounting-token type is set to the value MQACTT_UNKNOWN. In these environments use the *PutApplType* field to deduce the type of accounting token received.

- All other bytes are set to binary zero.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. If neither MQPMO_SET_IDENTITY_CONTEXT nor MQPMO_SET_ALL_CONTEXT is specified, this field is ignored on input and is an output-only field. For more information on message context, see the *WebSphere MQ Application Programming Guide*.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *AccountingToken* that was transmitted with the message if it was put to a queue. This will be the value of *AccountingToken* that is kept with the message if it is retained (see description of MQPMO_RETAIN in “Options (MQLONG)” on page 280 for more details about retained publications) but is not used as the *AccountingToken* when the message is sent as a publication to subscribers since they provide a value to override *AccountingToken* in all publications sent to them. 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.

You can use the following special value for the *AccountingToken* field:

MQACT_NONE

No accounting token is specified.

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

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

The length of this field is given by MQ_ACCOUNTING_TOKEN_LENGTH. The initial value of this field is MQACT_NONE.

ApplIdentityData (MQCHAR32)

This is part of the **identity context** of the message. For more information about message context, see “Overview for MQMD” on page 182; also see the *WebSphere MQ Application Programming Guide*.

ApplIdentityData is information that is defined by the application suite, and can be used to provide additional information about the message or its originator. The queue manager treats this information as character data, but does not define the format of it. When the queue manager generates this information, it is entirely blank.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. If a null character is present, the null and any following characters are converted to blanks by the queue manager. If neither

MQPMO_SET_IDENTITY_CONTEXT nor MQPMO_SET_ALL_CONTEXT is specified, this field is ignored on input and is an output-only field. For more information on message context, see the *WebSphere MQ Application Programming Guide*.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *ApplIdentityData* that was transmitted with the message if it was put to a queue. This will be the value of *ApplIdentityData* that is kept with the message if it is retained (see description of MQPMO_RETAIN for more details about retained publications) but is not used as the *ApplIdentityData* when the message is sent as a publication to subscribers since they provide a value to override *ApplIdentityData* in all publications sent to them. If the message has no context, the field is entirely blank.

This is an output field for the MQGET call. The length of this field is given by MQ_APPL_IDENTITY_DATA_LENGTH. The initial value of this field is the null string in C, and 32 blank characters in other programming languages.

ApplOriginData (MQCHAR4)

This is part of the **origin context** of the message. For more information about message context, see “Overview for MQMD” on page 182; also see the *WebSphere MQ Application Programming Guide*.

ApplOriginData is information that is defined by the application suite that can be used to provide additional information about the origin of the message. For example, it could be set by applications running with suitable user authority to indicate whether the identity data is trusted.

The queue manager treats this information as character data, but does not define the format of it. When the queue manager generates this information, it is entirely blank.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. Any information following a null character within the field is discarded. The queue manager converts the null character and any following characters to blanks. If MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

This is an output field for the MQGET call. The length of this field is given by MQ_APPL_ORIGIN_DATA_LENGTH. The initial value of this field is the null string in C, and 4 blank characters in other programming languages.

BackoutCount (MQLONG)

This is a count of the number of times that the message has been previously returned by the MQGET call as part of a unit of work, and subsequently backed out. It helps the application to detect processing errors that are based on message content. The count excludes MQGET calls that specify any of the MQGMO_BROWSE_* options.

The accuracy of this count is affected by the *HardenGetBackout* queue attribute; see “Attributes for queues” on page 615.

On z/OS, a value of 255 means that the message has been backed out 255 or more times; the value returned is never greater than 255.

This is an output field for the MQGET call. It is ignored for the MQPUT and MQPUT1 calls. The initial value of this field is 0.

CodedCharSetId (MQLONG)

This field specifies the character set identifier of character data within the message body.

Note: Character data in MQMD and the other MQ data structures that are parameters on calls must be in the character set of the queue manager. This is defined by the queue manager's *CodedCharSetId* attribute; see "Attributes for the queue manager" on page 575 for details of this attribute.

You must not use the following special value:

MQCCSI_APPL

This results in an incorrect value in the *CodedCharSetId* field of the MQMD and causes a return code of MQRC_SOURCE_CCSID_ERROR (or MQRC_FORMAT_ERROR for z/OS) when the message is received using the MQGET call with the MQGMO_CONVERT option.

You can use the following special values:

MQCCSI_Q_MGR

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

On the MQPUT and MQPUT1 calls, the queue manager changes this value in the MQMD that is sent with the message to the true character set identifier of the queue manager. As a result, the value MQCCSI_Q_MGR is never returned by the MQGET call.

MQCCSI_DEFAULT

The *CodedCharSetId* of the data in the *String* field is defined by the *CodedCharSetId* field in the header structure that precedes the MQCFH structure, or by the *CodedCharSetId* field in the MQMD if the MQCFH is at the start of the message.

MQCCSI_INHERIT

Character data in the message is in the same character set as this structure; this is the queue manager's character set. (For MQMD only, MQCCSI_INHERIT has the same meaning as MQCCSI_Q_MGR).

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

Do not use MQCCSI_INHERIT if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

MQCCSI_EMBEDDED

Character data in the message is in a character set whose identifier is contained within the message data itself. There can be any number of character set identifiers embedded within the message data, applying to different parts of the data. This value must be used for PCF messages (with a format of MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF) that contain data in a mixture of character sets. Each MQCFST, MQCFSL,

and MQCFST structure contained within the PCF message must have an explicit character set identifier specified and not MQCCSI_DEFAULT.

If a message of format MQFMT_EMBEDDED_PCF is to contain data in a mixture of character sets, do not use MQCCSI_EMBEDDED. Instead set MQEPH_CCSSID_EMBEDDED in the Flags field in the MQEPH structure. This is equivalent to setting MQCCSI_EMBEDDED in the preceding structure. Each MQCFST, MQCFSL, and MQCFST structure contained within the PCF message must then have an explicit character set identifier specified and not MQCCSI_DEFAULT. For more information on the MQEPH structure, see “MQEPH – Embedded PCF header” on page 120.

Specify this value only on the MQPUT and MQPUT1 calls. If it is specified on the MQGET call, it prevents conversion of the message.

On the MQPUT and MQPUT1 calls, the queue manager changes the values MQCCSI_Q_MGR and MQCCSI_INHERIT in the MQMD that is sent with the message as described above, but does not change the MQMD specified on the MQPUT or MQPUT1 call. No other check is carried out on the value specified.

Applications that retrieve messages must compare this field against the value the application is expecting; if the values differ, the application might need to convert character data in the message.

If you specify the MQGMO_CONVERT option 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 to convert the message data if necessary. If conversion is successful or unnecessary, the value is unchanged (except that the value MQCCSI_Q_MGR or MQCCSI_INHERIT is converted to the actual value). If conversion is unsuccessful, the value after the MQGET call represents the coded character set identifier of the unconverted message that is returned to the application.

Otherwise, this is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQCCSI_Q_MGR.

CorrelId (MQBYTE24)

The CorrelId field is property in the message header that may be used to identify a specific message or group of messages.

This is a byte string that the application can use to relate one message to another, or to relate the message to other work that the application is performing. The correlation identifier is a permanent property of the message, and persists across restarts of the queue manager. Because the correlation identifier is a byte string and not a character string, the correlation identifier is *not* converted between character sets when the message flows from one queue manager to another.

For the MQPUT and MQPUT1 calls, the application can specify any value. The queue manager transmits this value with the message and delivers it to the application that issues the get request for the message.

If the application specifies MQPMO_NEW_CORREL_ID, the queue manager generates a unique correlation identifier ¹ which is sent with the message, and also returned to the sending application on output from the MQPUT or MQPUT1 call.

1. A correlation identifier generated by the queue manager consists of a 3-byte product identifier (AMQ or CSQ in either ASCII or EBCDIC), followed by one reserved byte and a product-specific implementation of a unique string. In WebSphere MQ this

| This generated correlation identifier is kept with the message if it is retained, and
| is used as the correlation identifier when the message is sent as a publication to
| subscribers who specify MQCI_NONE in the SubCorrelId field in the MQSD
| passed on the MQSUB call. See MQPMO options for more details about retained
| publications.

When the queue manager or a message channel agent generates a report message, it sets the *CorrelId* field in the way specified by the *Report* field of the original message, either MQRO_COPY_MSG_ID_TO_CORREL_ID or MQRO_PASS_CORREL_ID. Applications that generate report messages must also do this.

For the MQGET call, *CorrelId* is one of the five fields that can be used to select a particular message to be retrieved from the queue. See the description of the *MsgId* field for details of how to specify values for this field.

Specifying MQCI_NONE as the correlation identifier has the same effect as *not* specifying MQMO_MATCH_CORREL_ID, that is, *any* correlation identifier will match.

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

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

The following special values can be used:

MQCI_NONE

No correlation identifier is specified.

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

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

MQCI_NEW_SESSION

Message is the start of a new session.

This value is recognized by the CICS bridge as indicating the start of a new session, that is, the start of a new sequence of messages.

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

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

product-specific implementation string contains the first 12 characters of the queue-manager name, and a value derived from the system clock. All queue managers that can intercommunicate must therefore have names that differ in the first 12 characters to ensure that message identifiers are unique. The ability to generate a unique string also depends on 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 must 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.

Encoding (MQLONG)

This specifies the numeric encoding of numeric data in the message; it does not apply to numeric data in the MQMD structure itself. The numeric encoding defines the representation used for binary integers, packed-decimal integers, and floating-point numbers.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that the field is valid. The following special value is defined:

MQENC_NATIVE

The encoding is the default for the programming language and machine on which the application is running.

Note: The value of this constant depends on the programming language and environment. For this reason, applications must be compiled using the header, macro, COPY, or INCLUDE files appropriate to the environment in which the application will run.

Applications that put messages usually specify MQENC_NATIVE. Applications that retrieve messages must compare this field against the value MQENC_NATIVE; if the values differ, the application might need to convert numeric data in the message. Use the MQGMO_CONVERT option to request the queue manager to convert the message as part of the processing of the MQGET call. See Chapter 6, "Machine encodings," on page 667 for details of how the *Encoding* field is constructed.

If you specify the MQGMO_CONVERT option on the MQGET call, this field is an input/output field. The value specified by the application is the encoding to which to convert the message data if necessary. If conversion is successful or unnecessary, the value is unchanged. If conversion is unsuccessful, the value after the MQGET call represents the encoding of the unconverted message that is returned to the application.

In other cases, this is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQENC_NATIVE.

Expiry (MQLONG)

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 that the message spends on the destination queue, and also on any intermediate transmission queues if the put is to a remote queue. It can also be decremented by message channel agents to reflect transmission times, if these are significant. Likewise, an application forwarding this message to another queue might decrement the value if necessary, if it has retained the message for a significant time. However, the expiration time is treated as approximate, and the value need not be decremented to reflect small time intervals.

When the message is retrieved by an application using the MQGET call, the *Expiry* field represents the amount of the original expiry time that still remains.

After a message's expiry time has elapsed, it becomes eligible to be discarded by the queue manager. The message is discarded when a browse or nonbrowse MQGET call occurs that would have returned the message had it not already expired. For example, a nonbrowse MQGET call with the *MatchOptions* field in MQGMO set to MQMO_NONE reading from a FIFO ordered queue discards all the expired messages up to the first unexpired message. With a priority ordered queue, the same call will discard expired messages of higher priority and messages of an equal priority that arrived on the queue before the first unexpired message.

A message that has expired is never returned to an application (either by a browse or a non-browse MQGET call), so the value in the *Expiry* field of the message descriptor after a successful MQGET call is either greater than zero, or the special value MQEL_UNLIMITED.

If a message is put on a remote queue, the message might expire (and be discarded) while it is on an intermediate transmission queue, before the message reaches the destination queue.

A report is generated when an expired message is discarded, if the message specified one of the MQRO_EXPIRATION_* report options. If none of these options is specified, no such report is generated; the message is assumed to be no longer relevant after this time period (perhaps because a later message has superseded it).

Any other program that discards messages based on expiry time must also send an appropriate report message if one was requested.

Note:

1. If a message is put with an *Expiry* time of zero or a number greater than 999 999 999, the MQPUT or MQPUT1 call fails with reason code MQRC_EXPIRY_ERROR; no report message is generated in this case.
2. Because a message whose expiry time has elapsed might not be discarded until later, there might be messages on a queue that have passed their expiry time, and that are not therefore eligible for retrieval. These messages nevertheless count toward 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 discarded, not when it becomes eligible for discarding.
4. Discarding an expired message, and generating 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 might become eligible to be discarded before it can be retrieved again.
6. If a nearly-expired message is locked by an MQGET call with MQGMO_LOCK, the message might become eligible to be discarded before it can be retrieved by an MQGET call with MQGMO_MSG_UNDER_CURSOR; reason code MQRC_NO_MSG_UNDER_CURSOR is returned on this subsequent MQGET call if that happens.
7. When a request message with an expiry time greater than zero is retrieved, the application can take one of the following actions when it sends the reply message:

- Copy the remaining expiry time from the request message to the reply message.
- Set the expiry time in the reply message to an explicit value greater than zero.
- Set the expiry time in the reply message to MQEI_UNLIMITED.

The action to take depends on the design of the application. However, the default action for putting messages to a dead-letter (undelivered-message) queue must be to preserve the remaining expiry time of the message, and to continue to decrement it.

8. Trigger messages are always generated with MQEI_UNLIMITED.
9. A message (normally on a transmission queue) that has a *Format* name of MQFMT_XMIT_Q_HEADER has a second message descriptor within the MQXQH. It therefore has two *Expiry* fields associated with it. The following additional points should be noted in this case:
 - When an application puts a message on a remote queue, the queue manager places the message initially on a local transmission queue, and prefixes the application message data with an MQXQH structure. The queue manager sets the values of the two *Expiry* fields to be the same as that specified by the application.

If an application puts a message directly on a local transmission queue, the message data must already begin with an MQXQH structure, and the format name must be MQFMT_XMIT_Q_HEADER. In this case, the application need not set the values of these two *Expiry* fields to be the same. (The queue manager checks that the *Expiry* field within the MQXQH contains a valid value, and that the message data is long enough to include it). For an application that can write directly to the transmission queue, the application has to create a transmission queue header with the embedded message descriptor. However, if the expiry value in the message descriptor written to the transmission queue is inconsistent with the value in the embedded message descriptor, an expiry error rejection occurs.

 - When a message with a *Format* name of MQFMT_XMIT_Q_HEADER is retrieved from a queue (whether this is a normal or a transmission queue), the queue manager decrements *both* these *Expiry* fields with the time spent waiting on the queue. No error is raised if the message data is not long enough to include the *Expiry* field in the MQXQH.
 - The queue manager uses the *Expiry* field in the separate message descriptor (that is, not the one in the message descriptor embedded within the MQXQH structure) to test whether the message is eligible for discarding.
 - If the initial values of the two *Expiry* fields are different, the *Expiry* time in the separate message descriptor when the message is retrieved might be greater than zero (so the message is not eligible for discarding), while the time according to the *Expiry* field in the MQXQH has elapsed. In this case the *Expiry* field in the MQXQH is set to zero.
10. The expiry time on a reply message returned from the IMS bridge is unlimited unless MQIIH_PASS_EXPIRATION is set in the Flags field of the MQIIH. See Flags for more information.

The following special value is recognized:

MQEI_UNLIMITED

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 MQEI_UNLIMITED.

Expired messages on z/OS:

On WebSphere MQ for z/OS, messages that have expired are discarded by the next appropriate MQGET call.

However, if no such call occurs, the expired message is not discarded, and, for some queues, a large number of expired messages can accumulate. To remedy this, set the queue manager to scan queues periodically and discard expired messages on one or more queues in one of the following ways:

Periodic scan

You can specify a period using the EXPRYINT (expiry interval) queue manager attribute. Each time the expiry interval is reached, the queue manager looks for candidate queues that are worth scanning to discard expired messages.

The queue manager maintains information about the expired messages on each queue, and knows whether a scan for expired messages is worthwhile. So, only a selection of queues is scanned at any time.

Shared queues are scanned by only one queue manager in a queue-sharing group. Generally, it is the first queue manager to restart, or the first to have EXPRYINT set. If this queue manager terminates, another queue manager in the queue-sharing group takes over the queue scanning. Set the expiry interval value for all queue managers within a queue-sharing group to the same value.

Explicit request

Issue the REFRESH QMGR TYPE(EXPIRY) command, specifying the queue or queues that you want scanned.

Feedback (MQLONG)

The Feedback field is used with a message of type MQMT_REPORT to indicate the nature of the report, and is only meaningful with that type of message.

The field can contain one of the MQFB_* values, or one of the MQRC_* values. Feedback codes are grouped as follows:

MQFB_NONE

No feedback provided.

MQFB_SYSTEM_FIRST

Lowest value for system-generated feedback.

MQFB_SYSTEM_LAST

Highest value for system-generated feedback.

The range of system-generated feedback codes MQFB_SYSTEM_FIRST through MQFB_SYSTEM_LAST includes the general feedback codes listed below (MQFB_*), and also the reason codes (MQRC_*) that can occur when the message cannot be put on the destination queue.

MQFB_APPL_FIRST

Lowest value for application-generated feedback.

MQFB_APPL_LAST

Highest value for application-generated feedback.

Applications that generate report messages must not use feedback codes in the system range (other than MQFB_QUIT), unless they want to simulate report messages generated by the queue manager or message channel agent.

On the MQPUT or MQPUT1 calls, the value specified must either be MQFB_NONE, or be within the system range or application range. This is checked whatever the value of *MsgType*.

General feedback codes:

MQFB_COA

Confirmation of arrival on the destination queue (see MQRO_COA).

MQFB_COD

Confirmation of delivery to the receiving application (see MQRO_COD).

MQFB_EXPIRATION

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

MQFB_PAN

Positive action notification (see MQRO_PAN).

MQFB_NAN

Negative action notification (see MQRO_NAN).

MQFB_QUIT

End application.

This can be used by a workload scheduling program to control the number of instances of an application program that are running. Sending an MQMT_REPORT message with this feedback code to an instance of the application program indicates to that instance that it should stop processing. However, adherence to this convention is a matter for the application; it is not enforced by the queue manager.

Channel feedback codes:

MQFB_CHANNEL_COMPLETED

A channel ended normally.

MQFB_CHANNEL_FAIL

A channel ended abnormally and will go into STOPPED state.

MQFB_CHANNEL_FAIL_RETRY

A channel ended abnormally and will go into RETRY state.

IMS-bridge feedback codes: When the IMS bridge receives a nonzero IMS-OTMA sense code, the IMS bridge converts the sense code from hexadecimal to decimal, adds the value MQFB_IMS_ERROR (300), and places the result in the *Feedback* field of the reply message. This results in the feedback code having a value in the range MQFB_IMS_FIRST (301) through MQFB_IMS_LAST (399) when an IMS-OTMA error has occurred.

The following feedback codes can be generated by the IMS bridge:

MQFB_DATA_LENGTH_ZERO

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

MQFB_DATA_LENGTH_NEGATIVE

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

MQFB_DATA_LENGTH_TOO_BIG

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

MQFB_BUFFER_OVERFLOW

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

MQFB_LENGTH_OFF_BY_ONE

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

MQFB_IIH_ERROR

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

MQFB_NOT_AUTHORIZED_FOR_IMS

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

MQFB_IMS_ERROR

An unexpected error was returned by IMS. Consult the WebSphere MQ error log on the system on which the IMS bridge resides for more information about the error.

MQFB_IMS_FIRST

IMS-generated feedback codes occupy the range MQFB_IMS_FIRST (300) through MQFB_IMS_LAST (399). The IMS-OTMA sense code itself is *Feedback* minus MQFB_IMS_ERROR.

MQFB_IMS_LAST

Highest value for IMS-generated feedback.

CICS-bridge feedback codes: The following feedback codes can be generated by the CICS bridge:

MQFB_CICS_APPL_ABENDED

The application program specified in the message abended. This feedback code occurs only in the *Reason* field of the MQDLH structure.

MQFB_CICS_APPL_NOT_STARTED

The EXEC CICS LINK for the application program specified in the message failed. This feedback code occurs only in the *Reason* field of the MQDLH structure.

MQFB_CICS_BRIDGE_FAILURE

CICS bridge terminated abnormally without completing normal error processing.

MQFB_CICS_CCSID_ERROR

Character set identifier not valid.

MQFB_CICS_CIH_ERROR

CICS information header structure missing or not valid.

MQFB_CICS_COMMAREA_ERROR

Length of CICS commarea not valid.

MQFB_CICS_CORREL_ID_ERROR

Correlation identifier not valid.

MQFB_CICS_DLQ_ERROR

The CICS bridge task was unable to copy a reply to this request to the dead-letter queue. The request was backed out.

MQFB_CICS_ENCODING_ERROR

Encoding not valid.

MQFB_CICS_INTERNAL_ERROR

CICS bridge encountered an unexpected error.

This feedback code occurs only in the *Reason* field of the MQDLH structure.

MQFB_CICS_NOT_AUTHORIZED

User identifier not authorized or password not valid.

This feedback code occurs only in the *Reason* field of the MQDLH structure.

MQFB_CICS_UOW_BACKED_OUT

The unit of work was backed out, for one of the following reasons:

- A failure was detected while processing another request within the same unit of work.
- A CICS abend occurred while the unit of work was in progress.

MQFB_CICS_UOW_ERROR

Unit-of-work control field *UOWControl* not valid.

Trace-route message feedback codes:**MQFB_ACTIVITY**

Used in conjunction with the MQFMT_EMBEDDED_PCF format to allow the option of user data following activity reports.

MQFB_MAX_ACTIVITIES

Returned when the trace-route message is discarded because the number of activities the message has been involved in exceeds the maximum activities limit.

MQFB_NOT_FORWARDED

Returned when the trace-route message is discarded because it is about to be sent to a remote queue manager that does not support trace-route messages.

MQFB_NOT_DELIVERED

Returned when the trace-route message is discarded because it is about to be put on a local queue.

MQFB_UNSUPPORTED_FORWARDING

Returned when the trace-route message is discarded because a value in the forwarding parameter is unrecognized, and is in the rejected bit mask.

MQFB_UNSUPPORTED_DELIVERY

Returned when the trace-route message is discarded because a value in the delivery parameter is unrecognized, and is in the rejected bit mask.

MQ reason codes: For exception report messages, *Feedback* contains an MQ reason code. Among possible reason codes are:

MQRC_PUT_INHIBITED

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

MQRC_Q_FULL

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

MQRC_NOT_AUTHORIZED

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

MQRC_Q_SPACE_NOT_AVAILABLE

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

MQRC_PERSISTENT_NOT_ALLOWED

(2048, X'800') Queue does not support persistent messages.

MQRC_MSG_TOO_BIG_FOR_Q_MGR

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

MQRC_MSG_TOO_BIG_FOR_Q

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

For a full list of reason codes, see:

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

.

This is an output field for the MQGET call, and an input field for MQPUT and MQPUT1 calls. The initial value of this field is MQFB_NONE.

Format (MQCHAR8)

This is a name that the sender of the message uses to indicate to the receiver the nature of the data in the message. Any characters that are in the queue manager's character set can 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 might not be possible to translate the name between the character sets of the sending and receiving queue managers.

Pad the name with blanks to the length of the field, or use a null character to terminate the name before the end of the field; the null and any subsequent characters are treated as blanks. Do not specify a name with leading or embedded blanks. For the MQGET call, the queue manager returns the name padded with blanks to the length of the field.

The queue manager does not check that the name complies with the recommendations described above.

Names beginning MQ in upper, lower, and mixed case have meanings that are defined by the queue manager; do not use names beginning with these letters for your own formats. The queue manager built-in formats are:

MQFMT_NONE

The nature of the data is undefined: the data cannot be converted when the message is retrieved from a queue using the MQGMO_CONVERT option.

If you specify MQGMO_CONVERT on the MQGET call, and the character set or encoding of data in the message differs from that specified in the

MsgDesc parameter, the message is returned with the following completion and reason codes (assuming no other errors):

- Completion code MQCC_WARNING and reason code MQRC_FORMAT_ERROR if the MQFMT_NONE data is at the beginning of the message.
- Completion code MQCC_OK and reason code MQRC_NONE if the MQFMT_NONE data is at the end of the message (that is, preceded by one or more MQ header structures). The MQ header structures are converted to the requested character set and encoding in this case.

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

MQFMT_ADMIN

The message is a command-server request or reply message in programmable command format (PCF). Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call. Refer to *WebSphere MQ Programmable Command Formats and Administration Interface* for more information about using programmable command format messages.

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

MQFMT_CICS

The message data begins with the CICS information header MQCIH, followed by the application data. The format name of the application data is given by the *Format* field in the MQCIH structure.

On z/OS, specify the MQGMO_CONVERT option on the MQGET call to convert messages that have format MQFMT_CICS.

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

MQFMT_COMMAND_1

The message is an MQSC command-server reply message containing the object count, completion code, and reason code. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_COMMAND_1_ARRAY is also defined; this has the same value as MQFMT_COMMAND_1, but is an array of characters instead of a string.

MQFMT_COMMAND_2

The message is an MQSC command-server reply message containing information about the objects requested. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_COMMAND_2_ARRAY is also defined; this has the same value as MQFMT_COMMAND_2, but is an array of characters instead of a string.

MQFMT_DEAD_LETTER_HEADER

The message data begins with the dead-letter header MQDLH. The data from the original message immediately follows the MQDLH structure. The format name of the original message data is given by the *Format* field in the MQDLH structure; see “MQDLH – Dead-letter header” on page 106 for details of this structure. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

COA and COD reports are not generated for messages that have a *Format* of MQFMT_DEAD_LETTER_HEADER.

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

MQFMT_DIST_HEADER

The message data begins with the distribution-list header MQDH; this includes the arrays of MQOR and MQPMR records. The distribution-list header can be followed by additional data. The format of the additional data (if any) is given by the *Format* field in the MQDH structure; see “MQDH – Distribution header” on page 99 for details of this structure. Messages with format MQFMT_DIST_HEADER can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

This format is supported in the following environments: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

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

MQFMT_EMBEDDED_PCF

Format for a trace-route message, provided that the PCF command value is set to MQCMD_TRACE_ROUTE. Using this format allows user data to be sent along with the trace-route message, provided that their applications can cope with preceding PCF parameters.

The PCF header **must** be the first header, or the message will not be treated as a trace-route message. This means that the message cannot be in a group, and that trace-route messages cannot be segmented. If a trace-route message is sent in a group the message is rejected with reason code MQRC_MSG_NOT_ALLOWED_IN_GROUP.

Note that MQFMT_ADMIN can also be used for the format of a trace-route message, but in this case no user data can be sent along with the trace-route message.

MQFMT_EVENT

The message is an MQ event message that reports an event that occurred. Event messages have the same structure as programmable commands; refer to the *WebSphere MQ Programmable Command Formats and Administration Interface* book for more information about this structure, and to the *WebSphere MQ Monitoring* book for information about events.

Version-1 event messages can be converted in all environments if the MQGMO_CONVERT option is specified on the MQGET call. Version-2 event messages can be converted only on z/OS.

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

MQFMT_IMS

The message data begins with the IMS information header `MQIIH`, which is followed by the application data. The format name of the application data is given by the *Format* field in the `MQIIH` structure.

Specify the `MQGMO_CONVERT` option on the `MQGET` call to convert messages that have format `MQFMT_IMS`.

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

MQFMT_IMS_VAR_STRING

The message is an IMS variable string, which is a string of the form `11zzccc`, where:

- 11** is a 2-byte length field specifying the total length of the IMS variable string item. This length is equal to the length of `11` (2 bytes), plus the length of `zz` (2 bytes), plus the length of the character string itself. `11` is a 2-byte binary integer in the encoding specified by the *Encoding* field.
- zz** is a 2-byte field containing flags that are significant to IMS. `zz` is a byte string consisting of two `MQBYTE` fields, and is transmitted without change from sender to receiver (that is, `zz` is not subject to any conversion).
- ccc** is a variable-length character string containing 1-4 characters. `ccc` is in the character set specified by the *CodedCharSetId* field.

On *z/OS*, the message data can consist of a sequence of IMS variable strings butted together, with each string being of the form `11zzccc`. There must be no bytes skipped between successive IMS variable strings. This means that if the first string has an odd length, the second string will be misaligned, that is, it will not begin on a boundary that is a multiple of two. Take care when constructing such strings on machines that require alignment of elementary data types.

Use the `MQGMO_CONVERT` option on the `MQGET` call to convert messages that have format `MQFMT_IMS_VAR_STRING`.

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

MQFMT_MD_EXTENSION

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 that follow the `MQMDE` are given by the *Format*, *CodedCharSetId*, and *Encoding* fields in the `MQMDE`. See “`MQMDE – Message descriptor extension`” on page 239 for details of this structure. Messages of this format can be converted if the `MQGMO_CONVERT` option is specified on the `MQGET` call.

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

MQFMT_PCF

The message is a user-defined message that conforms to the structure of a programmable command format (PCF) message. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call. Refer to the *WebSphere MQ Programmable Command Formats and Administration Interface* book for more information about using programmable command format messages.

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

MQFMT_REF_MSG_HEADER

The message data begins with the reference message header MQRMH, and is optionally followed by other data. The format name, character set, and encoding of the data is given by the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQRMH. See “MQRMH – Reference message header” on page 315 for details of this structure. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

This format is supported in the following environments: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

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

MQFMT_RF_HEADER

The message data begins with the rules and formatting header MQRFH, and is optionally followed by other data. The format name, character set, and encoding of the data (if any) are given by the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQRFH. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

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

MQFMT_RF_HEADER_2

The message data begins with the version-2 rules and formatting header MQRFH2, and is optionally followed by other data. The format name, character set, and encoding of the optional data (if any) are given by the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQRFH2. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_RF_HEADER_2_ARRAY is also defined; this has the same value as MQFMT_RF_HEADER_2, but is an array of characters instead of a string.

MQFMT_STRING

The application message data can be either an SBCS string (single-byte character set), or a DBCS string (double-byte character set). Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

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

MQFMT_TRIGGER

The message is a trigger message, described by the MQTM structure; see “MQTM – Trigger message” on page 366 for details of this structure. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

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

MQFMT_WORK_INFO_HEADER

The message data begins with the work information header MQWIH, which is followed by the application data. The format name of the application data is given by the *Format* field in the MQWIH structure.

On z/OS, specify the MQGMO_CONVERT option on the MQGET call to convert the *user data* in messages that have format MQFMT_WORK_INFO_HEADER. However, the MQWIH structure itself is always returned in the queue-manager’s character set and encoding (that is, the MQWIH structure is converted whether or not the MQGMO_CONVERT option is specified).

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

MQFMT_XMIT_Q_HEADER

The message data begins with the transmission queue header MQXQH. The data from the original message immediately follows the MQXQH structure. The format name of the original message data is given by the *Format* field in the MQMD structure, which is part of the transmission queue header MQXQH. See “MQXQH – Transmission-queue header” on page 387 for details of this structure.

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

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

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

GroupId (MQBYTE24)

This is a byte string that is used to identify the particular message group or logical message to which the physical message belongs. *GroupId* is also used if

segmentation is allowed for the message. In all these cases, *GroupId* has a non-null value, and one or more of the following flags is set in the *MsgFlags* field:

- MQMF_MSG_IN_GROUP
- MQMF_LAST_MSG_IN_GROUP
- MQMF_SEGMENT
- MQMF_LAST_SEGMENT
- MQMF_SEGMENTATION_ALLOWED

If none of these flags is set, *GroupId* has the special null value MQGI_NONE.

The application does not need to set this field on the MQPUT or MQGET call if:

- On the MQPUT call, MQPMO_LOGICAL_ORDER is specified.
- On the MQGET call, MQMO_MATCH_GROUP_ID is *not* specified.

These are the recommended ways of using these calls for messages that are not report messages. However, if the application requires more control, or the call is MQPUT1, the application must ensure that *GroupId* is set to an appropriate value.

Message groups and segments can be processed correctly only if the group identifier is unique. For this reason, *applications must not generate their own group identifiers*; instead, applications must do one of the following:

- If MQPMO_LOGICAL_ORDER is specified, the queue manager automatically generates a unique group identifier for the first message in the group or segment of the logical message, and uses that group identifier for the remaining messages in the group or segments of the logical message, so the application does not need to take any special action. This is the recommended procedure.
- If MQPMO_LOGICAL_ORDER is *not* specified, the application must request the queue manager to generate the group identifier, by setting *GroupId* to MQGI_NONE on the first MQPUT or MQPUT1 call for a message in the group or segment of the logical message. The group identifier returned by the queue manager on output from that call must then be used for the remaining messages in the group or segments of the logical message. If a message group contains segmented messages, the same group identifier must be used for all segments and messages in the group.

When MQPMO_LOGICAL_ORDER is not specified, messages in groups and segments of logical messages can be put in any order (for example, in reverse order), but the group identifier must be allocated by the *first* MQPUT or MQPUT1 call that is issued for any of those messages.

On input to the MQPUT and MQPUT1 calls, the queue manager uses the value described in Table 61 on page 284. On output from the MQPUT and MQPUT1 calls, the queue manager sets this field to the value that was sent with the message if the object opened is a single queue and not a distribution list, but leaves it unchanged if the object opened is a distribution list. In the latter case, if the application needs to know the group identifiers generated, the application must provide MQPMR records containing the *GroupId* field.

On input to the MQGET call, the queue manager uses the value described in Table 39 on page 149. On output from the MQGET call, the queue manager sets this field to the value for the message retrieved.

The following special value is defined:

MQGI_NONE

No group identifier specified.

The value is binary zero for the length of the field. This is the value that is used for messages that are not in groups, not segments of logical messages, and for which segmentation is not allowed.

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

The length of this field is given by `MQ_GROUP_ID_LENGTH`. The initial value of this field is `MQGI_NONE`. This field is ignored if *Version* is less than `MQMD_VERSION_2`.

MsgFlags (MQLONG)

These are flags that specify attributes of the message, or control its processing. The flags are divided into the following categories:

- Segmentation flags
- Status flags

Segmentation flags: When a message is too big for a queue, an attempt to put the message on the queue usually fails. Segmentation is a technique whereby the queue manager or application splits the message into smaller pieces called segments, and places each segment on the queue as a separate physical message. The application that retrieves the message can either retrieve the segments one by one, or request the queue manager to reassemble the segments into a single message that is returned by the `MQGET` call. The latter is achieved by specifying the `MQGMO_COMPLETE_MSG` option on the `MQGET` call, and supplying a buffer that is big enough to accommodate the complete message. (See “MQGMO – Get-message options” on page 126 for details of the `MQGMO_COMPLETE_MSG` option.) A message can be segmented at the sending queue manager, at an intermediate queue manager, or at the destination queue manager.

You can specify one of the following to control the segmentation of a message:

MQMF_SEGMENTATION_INHIBITED

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 is binary zero. This is the default.

MQMF_SEGMENTATION_ALLOWED

This option allows the message to be broken into segments by the queue manager. If specified for a message that is already a segment, this option allows the segment to be broken into smaller segments.

`MQMF_SEGMENTATION_ALLOWED` can be set without either `MQMF_SEGMENT` or `MQMF_LAST_SEGMENT` being set.

- On z/OS, the queue manager does not support the segmentation of messages. If a message is too big for the queue, the `MQPUT` or `MQPUT1` call fails with reason code `MQRC_MSG_TOO_BIG_FOR_Q`. However, the `MQMF_SEGMENTATION_ALLOWED` option can still be specified, and allows the message to be segmented at a remote queue manager.

When the queue manager segments a message, the queue manager turns on the `MQMF_SEGMENT` flag in the copy of the `MQMD` that is sent with each segment, but does not alter the settings of these flags in the `MQMD` provided by the application on the `MQPUT` or `MQPUT1` call. For the last

segment in the logical message, the queue manager also turns on the MQMF_LAST_SEGMENT flag in the MQMD that is sent with the segment.

Note: Take care when putting messages with MQMF_SEGMENTATION_ALLOWED but without MQPMO_LOGICAL_ORDER. If the message is:

- Not a segment, and
- Not in a group, and
- Not being forwarded,

the application must reset the *GroupId* field to MQGI_NONE before *each* MQPUT or MQPUT1 call, so that the queue manager can generate a unique group identifier for each message. If this is not done, unrelated messages can have the same group identifier, which might lead to incorrect processing subsequently. See the descriptions of the *GroupId* field and the MQPMO_LOGICAL_ORDER option for more information about when to reset the *GroupId* field.

The queue manager splits messages into segments as necessary so that the segments (plus any required header data) fit on the queue. However, there is a lower limit for the size of a segment generated by the queue manager (see below), and only the last segment created from a message can be smaller than this limit. (The lower limit for the size of an application-generated segment is one byte.) Segments generated by the queue manager might be of unequal length. The queue-manager processes the message as follows:

- User-defined formats are split on boundaries that are multiples of 16 bytes; the queue manager does not generate segments that are smaller than 16 bytes (other than the last segment).
- Built-in formats other than MQFMT_STRING are split at points appropriate to the nature of the data present. However, the queue manager never splits a message in the middle of an MQ header structure. This means that a segment containing a single MQ header structure cannot be split further by the queue manager, and as a result the minimum possible segment size for that message is greater than 16 bytes.

The second or later segment generated by the queue manager begins with one of the following:

- An MQ header structure
- The start of the application message data
- Part of the way through the application message data
- MQFMT_STRING is split without regard for the nature of the data present (SBCS, DBCS, or mixed SBCS/DBCS). When the string is DBCS or mixed SBCS/DBCS, this might result in segments that cannot be converted from one character set to another (see below). The queue manager never splits MQFMT_STRING messages into segments that are smaller than 16 bytes (other than the last segment).
- The queue manager sets the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQMD of each segment to describe correctly the data present at the *start* of the segment; the format name is either the name of a built-in format, or the name of a user-defined format.
- The *Report* field in the MQMD of segments with *Offset* greater than zero is modified. For each report type, if the report option is MQRO_*_WITH_DATA, but the segment cannot contain any of the first

100 bytes of user data (that is, the data following any MQ header structures that may be present), the report option is changed to MQRO_*.

The queue manager follows the above rules, but otherwise splits messages as it thinks fit; you cannot assume that the queue manager splits a message in a particular way.

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 during 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. 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 exists, the queue manager cannot perform segmentation. If the message does not require segmentation, the call can still succeed. But if the message requires segmentation, the call fails with reason code MQRC_UOW_NOT_AVAILABLE.

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

Take special care when converting data in messages that might be segmented:

- If the receiving application converts data on the MQGET call, and specifies the MQGMO_COMPLETE_MSG option, the data-conversion exit is passed the complete message for the exit to convert, and the fact that the message was segmented is be apparent to the exit.
- If the receiving application retrieves one segment at a time, the data-conversion exit is 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 might result in segments that cannot be converted by the exit, or the format is MQFMT_STRING and the character set is DBCS or mixed SBCS/DBCS, the sending application must create and put the segments, specifying MQMF_SEGMENTATION_INHIBITED to suppress further segmentation. In this way, the sending application can ensure that each segment contains sufficient information to allow the data-conversion exit to convert the segment successfully.

- If sender conversion is specified for a sending message channel agent (MCA), the MCA converts only messages that are not segments of logical messages; the MCA never attempts to convert messages that are segments.

This flag is an input flag on the MQPUT and MQPUT1 calls, and an output flag on the MQGET call. On the latter call, the queue manager also echoes the value of the flag to the *Segmentation* field in MQGMO.

The initial value of this flag is MQMF_SEGMENTATION_INHIBITED.

Status flags: These are flags that indicate whether the physical message belongs to a message group, is a segment of a logical message, both, or neither. One or more of the following can be specified on the MQPUT or MQPUT1 call, or returned by the MQGET call:

MQMF_MSG_IN_GROUP

Message is a member of a group.

MQMF_LAST_MSG_IN_GROUP

Message is the last logical message in a group.

If this flag is set, the queue manager turns on MQMF_MSG_IN_GROUP in the copy of MQMD that is sent with the message, but does not alter the settings of these flags in the MQMD provided by the application on the MQPUT or MQPUT1 call.

It is valid for a group to consist of only one logical message. If this is the case, MQMF_LAST_MSG_IN_GROUP is set, but the *MsgSeqNumber* field has the value one.

MQMF_SEGMENT

Message is a segment of a logical message.

When MQMF_SEGMENT is specified without MQMF_LAST_SEGMENT, the length of the application message data in the segment (*excluding* the lengths of any MQ header structures that might be present) must be at least one. If the length is zero, the MQPUT or MQPUT1 call fails with reason code MQRC_SEGMENT_LENGTH_ZERO.

On z/OS, this option is not supported if the message is being put on a queue that has an index type of MQIT_GROUP_ID.

MQMF_LAST_SEGMENT

Message is the last segment of a logical message.

If this flag is set, the queue manager turns on MQMF_SEGMENT in the copy of MQMD that is sent with the message, but does not alter the settings of these flags in the MQMD provided by the application on the MQPUT or MQPUT1 call.

A logical message can consist of only one segment. If this is the case, MQMF_LAST_SEGMENT is set, but the *Offset* field has the value zero.

When MQMF_LAST_SEGMENT is specified, the length of the application message data in the segment (*excluding* the lengths of any header structures that might be present) can be zero.

On z/OS, this option is not supported if the message is being put on a queue that has an index type of MQIT_GROUP_ID.

The application must ensure that these flags are set correctly when putting messages. If MQPMO_LOGICAL_ORDER is specified, or was specified on the preceding MQPUT call for the queue handle, the settings of the flags must be consistent with the group and segment information retained by the queue manager for the queue handle. The following conditions apply to *successive* MQPUT calls for the queue handle when MQPMO_LOGICAL_ORDER is specified:

- If there is no current group or logical message, all these flags (and combinations of them) are valid.

- Once MQMF_MSG_IN_GROUP has been specified, it must remain on until MQMF_LAST_MSG_IN_GROUP is specified. The call fails with reason code MQRC_INCOMPLETE_GROUP if this condition is not satisfied.
- Once MQMF_SEGMENT has been specified, it must remain on until MQMF_LAST_SEGMENT is specified. The call fails with reason code MQRC_INCOMPLETE_MSG if this condition is not satisfied.
- Once MQMF_SEGMENT has been specified without MQMF_MSG_IN_GROUP, MQMF_MSG_IN_GROUP must remain *off* until after MQMF_LAST_SEGMENT has been specified. The call fails with reason code MQRC_INCOMPLETE_MSG if this condition is not satisfied.

Table 61 on page 284 shows the valid combinations of the flags, and the values used for various fields.

These flags are input flags on the MQPUT and MQPUT1 calls, and output flags on the MQGET call. On the latter call, the queue manager also echoes the values of the flags to the *GroupStatus* and *SegmentStatus* fields in MQGMO.

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

MQMF_NONE

No message flags (default message attributes).

This inhibits segmentation, and indicates that the message is not in a group and is not a segment of a logical message. MQMF_NONE is defined to aid program documentation. It is not intended that this flag be used with any other, but as its value is zero, such use cannot be detected.

The *MsgFlags* field is partitioned into subfields; for details see Chapter 7, “Report options and message flags,” on page 671.

The initial value of this field is MQMF_NONE. This field is ignored if *Version* is less than MQMD_VERSION_2.

MsgId (MQBYTE24)

This is a byte string that is used to distinguish one message from another. Generally, no two messages should have the same message identifier, although this is not disallowed by the queue manager. The message identifier is a permanent property of the message, and persists across restarts of the queue manager. Because the message identifier is a byte string and not a character string, the message identifier is *not* converted between character sets when the message flows from one queue manager to another.

For the MQPUT and MQPUT1 calls, if MQMI_NONE or MQPMO_NEW_MSG_ID is specified by the application, the queue manager generates a unique message identifier² 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

2. A *MsgId* generated by the queue manager consists of a 4-byte product identifier (AMQb or CSQb in either ASCII or EBCDIC, where b represents a blank), followed by a product-specific implementation of a unique string. In WebSphere MQ this contains the first 12 characters of the queue-manager name, and a value derived from the system clock. All queue managers that can intercommunicate must therefore have names that differ in the first 12 characters, in order to ensure that message identifiers are unique. The ability to generate a unique string also depends on 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 must 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.

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 topic, the queue manager generates unique message identifiers as necessary for each message published. If MQPMO_NEW_MSG_ID is specified by the application, the queue manager generates a unique message identifier to return on output. The value of the *MsgId* field in the MQMD is unchanged on return from the call if MQMI_NONE or MQPMO_NEW_MSG_ID was specified.

See the description of MQPMO_RETAIN in “Options (MQLONG)” on page 280 for more details about retained publications.

If the message is being put to a distribution list, the queue manager generates unique message identifiers as necessary, but the value of the *MsgId* field in MQMD is unchanged on return from the call, even if MQMI_NONE or MQPMO_NEW_MSG_ID was specified. If the application needs to know the message identifiers generated by the queue manager, the application must provide MQPMR records containing the *MsgId* field.

The sending application can also specify a value for the message identifier other than MQMI_NONE; this stops the queue manager generating a unique message identifier. An application that is forwarding a message can use this to propagate the message identifier of the original message.

The queue manager does not use this field except to:

- Generate a unique value if requested, as described above
- Deliver the value to the application that issues the get request for the message
- Copy the value to the *CorrelId* field of any report message that it generates about this message (depending on the *Report* options)

When the queue manager or a message channel agent generates a report message, it sets the *MsgId* field in the way specified by the *Report* field of the original message, either MQRO_NEW_MSG_ID or MQRO_PASS_MSG_ID. Applications that generate report messages must also do this.

For the MQGET call, *MsgId* is one of the five fields that can be used to retrieve a particular message from the queue. Normally the MQGET call returns the next message on the queue, but a particular message can be obtained by specifying one or more of the five selection criteria, in any combination; these fields are:

- *MsgId*
- *CorrelId*
- *GroupId*
- *MsgSeqNumber*
- *Offset*

The application sets one or more of these field to the values required, and then sets the corresponding MQMO_* match options in the *MatchOptions* field in MQGMO to use those fields as selection criteria. Only messages that have the specified values in those fields are candidates for retrieval. The default for the *MatchOptions* field (if not altered by the application) is to match both the message identifier and the correlation identifier.

On z/OS, the selection criteria that you can use are restricted by the type of index used for the queue. See the *IndexType* queue attribute for further details.

Normally, the message returned is the *first* message on the queue that satisfies the selection criteria. But if MQGMO_BROWSE_NEXT is specified, the message returned is the *next* message that satisfies the selection criteria; the scan for this message starts with the message *following* the current cursor position.

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

- an MQGET call by *CorrelId* on 64-bit distributed platforms where the *CorrelId* index eliminates the need to perform a true sequential scan.
- an MQGET call by *IndexType* on z/OS.

In both these cases, retrieval performance is improved.

See Table 39 on page 149 for more information about how selection criteria are used in various situations.

Specifying MQMI_NONE as the message identifier has the same effect as *not* specifying MQMO_MATCH_MSG_ID, that is, *any* message identifier matches.

This field is ignored if the MQGMO_MSG_UNDER_CURSOR option is specified in the *GetMsgOpts* parameter on the MQGET call.

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

The following special value can be used:

MQMI_NONE

No message identifier is specified.

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

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

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

MsgSeqNumber (MQLONG)

This is the sequence number of a logical message within a 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 that is not in a group has a sequence number of 1.

The application does not have to set this field on the MQPUT or MQGET call if:

- On the MQPUT call, MQPMO_LOGICAL_ORDER is specified.
- On the MQGET call, MQMO_MATCH_MSG_SEQ_NUMBER is *not* specified.

These are the recommended ways of using these calls for messages that are not report messages. However, if the application requires more control, or the call is MQPUT1, the application must ensure that *MsgSeqNumber* is set to an appropriate value.

On input to the MQPUT and MQPUT1 calls, the queue manager uses the value described in Table 61 on page 284. 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 shown in Table 39 on page 149. On output from the MQGET call, the queue manager sets this field to the value for the message retrieved.

The initial value of this field is one. This field is ignored if *Version* is less than MQMD_VERSION_2.

MsgType (MQLONG)

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

MQMT_SYSTEM_FIRST

Lowest value for system-defined message types.

MQMT_SYSTEM_LAST

Highest value for system-defined message types.

The following values are currently defined within the system range:

MQMT_DATAGRAM

The message is one that does not require a reply.

MQMT_REQUEST

The message is one that requires a reply.

Specify the name of the queue to which to send the reply in the *ReplyToQ* field. The *Report* field indicates how to set the *MsgId* and *CorrelId* of the reply.

MQMT_REPLY

The message is the reply to an earlier request message (MQMT_REQUEST). The message must be sent to the queue indicated by the *ReplyToQ* field of the request message. Use the *Report* field of the request to control how to set the *MsgId* and *CorrelId* of the reply.

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

MQMT_REPORT

The message is reporting on some expected or unexpected occurrence, usually related to some other message (for example, a request message was received that contained data that was not valid). Send the message to the queue indicated by the *ReplyToQ* field of the message descriptor of the original message. Set the *Feedback* fields to indicate the nature of the report. Use the *Report* field of the original message to control how to set the *MsgId* and *CorrelId* of the report message.

Report messages generated by the queue manager or message channel agent are always sent to the *ReplyToQ* queue, with the *Feedback* and *CorrelId* fields set as described above.

Application-defined values can also be used. They must be within the following range:

MQMT_APPL_FIRST

Lowest value for application-defined message types.

MQMT_APPL_LAST

Highest value for application-defined message types.

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

This is an output field for the MQGET call, and an input field for MQPUT and MQPUT1 calls. The initial value of this field is MQMT_DATAGRAM.

Offset (MQLONG)

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 that is not a segment of a logical message has an offset of zero.

The application does not need to set this field on the MQPUT or MQGET call if:

- On the MQPUT call, MQPMO_LOGICAL_ORDER is specified.
- On the MQGET call, MQMO_MATCH_OFFSET is *not* specified.

These are the recommended ways of using these calls for messages that are not report messages. However, if the application does not comply with these conditions, or the call is MQPUT1, the application must ensure that *Offset* is set to an appropriate value.

On input to the MQPUT and MQPUT1 calls, the queue manager uses the value described in Table 61 on page 284. On output from the MQPUT and MQPUT1 calls, the queue manager sets this field to the value that was sent with the message.

For a report message reporting on a segment of a logical message, the *OriginalLength* field (provided it is not MQOL_UNDEFINED) is used to update the offset in the segment information retained by the queue manager.

On input to the MQGET call, the queue manager uses the value shown in Table 39 on page 149. On output from the MQGET call, the queue manager sets this field to the value for the message retrieved.

The initial value of this field is zero. This field is ignored if *Version* is less than MQMD_VERSION_2.

OriginalLength (MQLONG)

This field is relevant only for report messages that are segments. It specifies the length of the message segment to which the report message relates; it does not specify the length of the logical message of which the segment forms part, or the length of the data in the report message.

Note: When generating a report message for a message that is a segment, the queue manager and message channel agent copy into the MQMD for the report

message the *GroupId*, *MsgSeqNumber*, *Offset*, and *MsgFlags*, fields from the original message. As a result, the report message is also a segment. Applications that generate report messages must do the same, and set the *OriginalLength* field correctly.

The following special value is defined:

MQOL_UNDEFINED

Original length of message not defined.

OriginalLength is an input field on the MQPUT and MQPUT1 calls, but the value that the application provides is accepted only in particular circumstances:

- If the message being put is a segment and is also a report message, the queue manager accepts the value specified. The value must be:
 - Greater than zero if the segment is not the last segment
 - Not less than zero if the segment is the last segment
 - Not less than the length of data present in the message

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

- If the message being put is a segment but not a report message, the queue manager ignores the field and uses the length of the application message data instead.
- In all other cases, the queue manager ignores the field and uses the value MQOL_UNDEFINED instead.

This is an output field on the MQGET call.

The initial value of this field is MQOL_UNDEFINED. This field is ignored if *Version* is less than MQMD_VERSION_2.

Persistence (MQLONG)

This indicates whether the message survives system failures and restarts of the queue manager. For the MQPUT and MQPUT1 calls, the value must be one of the following:

MQPER_PERSISTENT

The message survives system failures and restarts of the queue manager. Once the message has been put, and the unit of work in which it was put has been committed (if the message is put as part of a unit of work), the message is preserved on auxiliary storage. It remains there until the message is removed from the queue, and the unit of work in which it was put has been committed (if the message is retrieved as part of a unit of work).

When a persistent message is sent to a remote queue, a store-and-forward mechanism holds the message at each queue manager along the route to the destination, until the message is known to have arrived at the next queue manager.

Persistent messages cannot be placed on:

- Temporary dynamic queues
- Shared queues that map to a CFSTRUCT object at CFLEVEL(2) or below, or where the CFSTRUCT object is defined as RECOVER(NO).

Persistent messages can be placed on permanent dynamic queues, and predefined queues.

MQPER_NOT_PERSISTENT

The message does not usually survive system failures or queue manager restarts. This applies even if an intact copy of the message is found on auxiliary storage when the queue manager restarts.

In the case of NPMCLASS (HIGH) queues nonpersistent messages survive a normal queue manager shutdown and restart.

In the case of shared queues, nonpersistent messages survive queue manager restarts in the queue-sharing group, but do not survive failures of the coupling facility used to store messages on the shared queues.

MQPER_PERSISTENCE_AS_Q_DEF

- If the queue is a cluster queue, the persistence of the message is taken from the *DefPersistence* attribute defined at the *destination* queue manager that owns the particular instance of the queue on which the message is placed. Usually, all instances of a cluster queue have the same value for the *DefPersistence* attribute, although this is not mandated.

The value of *DefPersistence* is copied into the *Persistence* field when the message is placed on the destination queue. If *DefPersistence* is changed subsequently, messages that have already been placed on the queue are not affected.

- If the queue is not a cluster queue, the persistence of the message is taken from the *DefPersistence* attribute defined at the *local* queue manager, even if the destination queue manager is remote.

If there is more than one definition in the queue-name resolution path, the default persistence is taken from the value of this attribute in the *first* definition in the path. This can be:

- An alias queue
- A local queue
- A local definition of a remote queue
- A queue-manager alias
- A transmission queue (for example, the *DefXmitQName* queue)

The value of *DefPersistence* is copied into the *Persistence* field when the message is put. If *DefPersistence* is changed subsequently, messages that have already been put are not affected.

Both persistent and nonpersistent messages can exist on the same queue.

When replying to a message, applications must use the persistence of the request message for the reply message.

For an MQGET call, the value returned is either MQPER_PERSISTENT or MQPER_NOT_PERSISTENT.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQPER_PERSISTENCE_AS_Q_DEF.

Priority (MQLONG)

For the MQPUT and MQPUT1 calls, the value must be greater than or equal to zero; zero is the lowest priority. The following special value can also be used:

MQPRI_PRIORITY_AS_Q_DEF

- If the queue is a cluster queue, the priority for the message is taken from the *DefPriority* attribute as defined at the *destination* queue manager that owns the particular instance of the queue on which the message is placed. Usually, all instances of a cluster queue have the same value for the *DefPriority* attribute, although this is not mandated.

The value of *DefPriority* is copied into the *Priority* field when the message is placed on the destination queue. If *DefPriority* is changed subsequently, messages that have already been placed on the queue are not affected.

- If the queue is not a cluster queue, the priority for the message is taken from the *DefPriority* attribute as defined at the *local* queue manager, even if the destination queue manager is remote.

If there is more than one definition in the queue-name resolution path, the default priority is taken from the value of this attribute in the *first* definition in the path. This can be:

- An alias queue
- A local queue
- A local definition of a remote queue
- A queue-manager alias
- A transmission queue (for example, the *DefXmitQName* queue)

The value of *DefPriority* is copied into the *Priority* field when the message is put. If *DefPriority* is changed subsequently, messages that have already been put are not affected.

The value returned by the MQGET call is always greater than or equal to zero; the value MQPRI_PRIORITY_AS_Q_DEF is never returned.

If a message is put with a priority greater than the maximum supported by the local queue manager (this maximum is given by the *MaxPriority* queue-manager attribute), the message is accepted by the queue manager, but placed on the queue at the queue manager's maximum priority; the MQPUT or MQPUT1 call completes with MQCC_WARNING and reason code MQRC_PRIORITY_EXCEEDS_MAXIMUM. However, the *Priority* field retains the value specified by the application that put the message.

On z/OS, if a message with a *MsgSeqNumber* of 1 is put to a queue that has a message delivery sequence of MQMDS_PRIORITY and an index type of MQIT_GROUP_ID, the queue might treat the message with a different priority. If the message was placed on the queue with a priority of 0 or 1, it is processed as though it has a priority of 2. This is because the order of messages placed on this type of queue is optimized to enable efficient group completeness tests. For more information on the message delivery sequence MQMDS_PRIORITY and the index type MQIT_GROUP_ID, see *MsgDeliverySequence* attribute.

When replying to a message, applications must use the priority of the request message for the reply message. In other situations, specifying MQPRI_PRIORITY_AS_Q_DEF allows priority tuning to be carried out without changing the application.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQPRI_PRIORITY_AS_Q_DEF.

PutAppName (MQCHAR28)

This is the name of application that put the message, and is part of the *origin context* of the message. The contents differ between platforms, and might differ between releases.

For more information about message context, see “Overview for MQMD” on page 182; also see the *WebSphere MQ Application Programming Guide*.

The format of *PutAppName* depends on the value of *PutApplType* and can change from one release to another. Changes are rare, but do happen if the environment changes.

When the queue manager sets this field (that is, for all options except MQPMO_SET_ALL_CONTEXT), it sets the field to a value that is determined by the environment:

- On z/OS, the queue manager uses:
 - For z/OS batch, the 8-character job name from the JES JOB card
 - For TSO, the 7-character TSO user identifier
 - For CICS, the 8-character applid, followed by the 4-character tranid
 - For IMS, the 8-character IMS system identifier, followed by the 8-character PSB name
 - For XCF, the 8-character XCF group name, followed by the 16-character XCF member name
 - For a message generated by a queue manager, the first 28 characters of the queue manager name
 - For distributed queuing without CICS, the 8-character jobname of the channel initiator followed by the 8-character name of the module putting to the dead-letter queue followed by an 8-character task identifier.

The name or names are each padded to the right with blanks, as is any space in the remainder of the field. Where there is more than one name, there is no separator between them.

- On Windows systems, the queue manager uses:
 - For a CICS application, the CICS transaction name
 - For a non-CICS application, the rightmost 28 characters of the fully-qualified name of the executable
- On i5/OS, the queue manager uses the fully-qualified job name.
- On UNIX systems, the queue manager uses:
 - For a CICS application, the CICS transaction name
 - For a non-CICS application, at least the rightmost 14 characters of the fully-qualified name of the executable, if this is available to the queue manager, and blanks otherwise.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. Any information following a null character within the field is discarded. The null character and any following characters are converted to blanks by the queue manager. If MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

PutApplType (MQLONG)

This is the type of application that put the message, and is part of the **origin context** of the message. For more information about message context, see “Overview for MQMD” on page 182; also see the *WebSphere MQ Application Programming Guide*.

PutApplType can have one of the following standard types. You can also define your own types, but only with values in the range MQAT_USER_FIRST through MQAT_USER_LAST.

MQAT_AIX

AIX application (same value as MQAT_UNIX).

MQAT_BROKER

Broker.

MQAT_CICS

CICS transaction.

MQAT_CICS_BRIDGE

CICS bridge.

MQAT_CICS_VSE

CICS/VSE[®] transaction.

MQAT_DOS

WebSphere MQ client application on PC DOS.

MQAT_DQM

Distributed queue manager agent.

MQAT_GUARDIAN

Tandem Guardian application (same value as MQAT_NSK).

MQAT_IMS

IMS application.

MQAT_IMS_BRIDGE

IMS bridge.

MQAT_JAVA

Java[™].

MQAT_MVS

MVS or TSO application (same value as MQAT_ZOS).

MQAT_NOTES_AGENT

Lotus Notes[®] Agent application.

MQAT_NSK

Compaq NonStop Kernel application.

MQAT_OS2

OS/2[®] or Presentation Manager application.

MQAT_OS390

OS/390[®] application (same value as MQAT_ZOS).

MQAT_OS400

i5/OS application.

MQAT_QMGR

Queue manager.

MQAT_UNIX
UNIX application.

MQAT_VMS
Digital OpenVMS application.

MQAT_VOS
Stratus VOS application.

MQAT_WINDOWS
16-bit Windows application.

MQAT_WINDOWS_NT
32-bit Windows application.

MQAT_WLM
z/OS workload manager application.

MQAT_XCF
XCF.

MQAT_ZOS
z/OS application.

MQAT_DEFAULT
Default application type.

This is the default application type for the platform on which the application is running.

Note: The value of this constant is environment-specific. Because of this, always compile the application using the header, include, or COPY files that are appropriate to the platform on which the application will run.

MQAT_UNKNOWN
Use this value to indicate that the application type is unknown, even though other context information is present.

MQAT_USER_FIRST
Lowest value for user-defined application type.

MQAT_USER_LAST
Highest value for user-defined application type.

The following special value can also occur:

MQAT_NO_CONTEXT
This value is set by the queue manager when a message is put with no context (that is, the MQPMO_NO_CONTEXT context option is specified).

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

When the queue manager generates this information as a result of an application put, the field is set to a value that is determined by the environment. On i5/OS, it is set to MQAT_OS400; the queue manager never uses MQAT_CICS on i5/OS.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. If MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

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

PutDate (MQCHAR8)

This is the date when the message was put, and is part of the **origin context** of the message. For more information about message context, see “Overview for MQMD” on page 182; also see the *WebSphere MQ Application Programming Guide*.

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

- YYYYMMDD

where the characters represent:

YYYY year (four numeric digits)

MM month of year (01 through 12)

DD day of month (01 through 31)

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

If the message was put as part of a unit of work, the date is that when the message was put, and not the date when the unit of work was committed.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. The contents of the field are not checked by the queue manager, except that any information following a null character within the field is discarded. The queue manager converts the null character and any following characters to blanks. If MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

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

PutTime (MQCHAR8)

This is the time when the message was put, and is part of the **origin context** of the message. For more information about message context, see “Overview for MQMD” on page 182; also see the *WebSphere MQ Application Programming Guide*.

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

- HHMMSSSTH

where the characters represent (in order):

HH hours (00 through 23)

MM minutes (00 through 59)

SS seconds (00 through 59; see note below)

- T tenths of a second (0 through 9)
- H hundredths of a second (0 through 9)

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

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

If the message was put as part of a unit of work, the time is that when the message was put, and not the time when the unit of work was committed.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. The queue manager does not check the contents of the field, except that any information following a null character within the field is discarded. The queue manager converts the null character and any following characters to blanks. If MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

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

ReplyToQ (MQCHAR48)

This is the name of the message queue to which the application that issued the get request for the message sends MQMT_REPLY and MQMT_REPORT messages. The name is the local name of a queue that is defined on the queue manager identified by *ReplyToQMgr*. This queue must not be a model queue, although the sending queue manager does not verify this when the message is put.

For the MQPUT and MQPUT1 calls, this field must not be blank if the *MsgType* field has the value MQMT_REQUEST, or if any report messages are requested by the *Report* field. However, the value specified (or substituted; see below) is passed on to the application that issues the get request for the message, whatever the message type.

If the *ReplyToQMgr* field is blank, the local queue manager looks up the *ReplyToQ* name in its own queue definitions. If a local definition of a remote queue exists with this name, the *ReplyToQ* value in the transmitted message is replaced by the value of the *RemoteQName* attribute from the definition of the remote queue, and this value is returned in the message descriptor when the receiving application issues an MQGET call for the message. If a local definition of a remote queue does not exist, *ReplyToQ* is unchanged.

If the name is specified, it can contain trailing blanks; the first null character and characters following it are treated as blanks. Otherwise no check is made that the name satisfies the naming rules for queues; this is also true for the name transmitted, if the *ReplyToQ* is replaced in the transmitted message. The only check made is that a name has been specified, if the circumstances require it.

If a reply-to queue is not required, set the *ReplyToQ* field to blanks, or (in the C programming language) to the null string, or to one or more blanks followed by a null character; do not leave the field uninitialized.

For the MQGET call, the queue manager always returns the name padded with blanks to the length of the field.

If a message that requires a report message cannot be delivered, and the report message also cannot be delivered to the queue specified, both the original message and the report message go to the dead-letter (undelivered-message) queue (see the *DeadLetterQName* attribute described in “Attributes for the queue manager” on page 575).

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ReplyToQMgr (MQCHAR48)

This is the name of the queue manager to which to send the reply message or report message. *ReplyToQ* is the local name of a queue that is defined on this queue manager.

If the *ReplyToQMgr* field is blank, the local queue manager looks up the *ReplyToQ* name in its queue definitions. If a local definition of a remote queue exists with this name, the *ReplyToQMgr* value in the transmitted message is replaced by the value of the *RemoteQMgrName* attribute from the definition of the remote queue, and this value is returned in the message descriptor when the receiving application issues an MQGET call for the message. If a local definition of a remote queue does not exist, the *ReplyToQMgr* that is transmitted with the message is the name of the local queue manager.

If the name is specified, it can contain trailing blanks; the first null character and characters following it are treated as blanks. Otherwise no check is made that the name satisfies the naming rules for queue managers, or that this name is known to the sending queue manager; this is also true for the name transmitted, if the *ReplyToQMgr* is replaced in the transmitted message. For more information about names, see the *WebSphere MQ Application Programming Guide*.

If a reply-to queue is not required, set the *ReplyToQMgr* field to blanks, or (in the C programming language) to the null string, or to one or more blanks followed by a null character; do not leave the field uninitialized.

For the MQGET call, the queue manager always returns the name padded with blanks to the length of the field.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The length of this field is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

Report (MQLONG)

A report message is a message about another message, used to inform an application about expected or unexpected events that relate to the original message. The *Report* field enables the application sending the original message to specify which report messages are required, whether the application message data is to be included in them, and also (for both reports and replies) how the message

and correlation identifiers in the report or reply message are to be set. Any or all (or none) of the following types of report message can be requested:

- Exception
- Expiration
- Confirm on arrival (COA)
- Confirm on delivery (COD)
- Positive action notification (PAN)
- Negative action notification (NAN)

If more than one type of report message is required, or other report options are needed, the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

The application that receives the report message can determine the reason that the report was generated by examining the *Feedback* field in the MQMD; see the *Feedback* field for more details.

The use of report options when putting a message to a topic can cause zero, one or many report messages to be generated and sent to the application. This is because the publication message may be sent to zero, one or many subscribing applications.

Exception options: Specify one of the options listed below to request an exception report message.

MQRO_EXCEPTION

A message channel agent generates this type of report when a message is sent to another queue manager and the message cannot be delivered to the specified destination queue. For example, the destination queue or an intermediate transmission queue might be full, or the message might be too big for the queue.

Generation of the exception report message depends on the persistence of the original message, and the speed of the message channel (normal or fast) through which the original message travels:

- For all persistent messages, and for nonpersistent messages traveling through normal message channels, the exception report is generated *only* if the action specified by the sending application for the error condition can be completed successfully. The sending application can specify one of the following actions to control the disposition of the original message when the error condition arises:
 - MQRO_DEAD_LETTER_Q (this places the original message on the dead-letter queue).
 - MQRO_DISCARD_MSG (this discards the original message).

If the action specified by the sending application cannot be completed successfully, the original message is left on the transmission queue, and no exception report message is generated.

- For nonpersistent messages traveling through fast message channels, the original message is removed from the transmission queue and the exception report generated *even if* the specified action for the error condition cannot be completed successfully. For example, if MQRO_DEAD_LETTER_Q is specified, but the original message cannot

be placed on the dead-letter queue because that queue is full, the exception report message is generated and the original message discarded.

Refer to the *WebSphere MQ Intercommunications* book for more information about normal and fast message channels.

An exception report is not generated if the application that put the original message can be notified synchronously of the problem by means of the reason code returned by the MQPUT or MQPUT1 call.

Applications can also send exception reports, to indicate that a message cannot be processed (for example, because it is a debit transaction that would cause the account to exceed its credit limit).

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

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

MQRO_EXCEPTION_WITH_DATA

This is the same as MQRO_EXCEPTION, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

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

MQRO_EXCEPTION_WITH_FULL_DATA

Exception reports with full data required.

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

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

Expiration options: Specify one of the options listed below to request an expiration report message.

MQRO_EXPIRATION

This type of report is generated by the queue manager if the message is discarded before delivery to an application because its expiry time has passed (see the *Expiry* field). If this option is not set, no report message is generated if a message is discarded for this reason (even if you specify one of the MQRO_EXCEPTION_* options).

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

Do not specify more than one of MQRO_EXPIRATION, MQRO_EXPIRATION_WITH_DATA, and MQRO_EXPIRATION_WITH_FULL_DATA.

MQRO_EXPIRATION_WITH_DATA

This is the same as MQRO_EXPIRATION, except that the first 100 bytes of the application message data from the original message are included in the

report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of MQRO_EXPIRATION, MQRO_EXPIRATION_WITH_DATA, and MQRO_EXPIRATION_WITH_FULL_DATA.

MQRO_EXPIRATION_WITH_FULL_DATA

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

Do not specify more than one of MQRO_EXPIRATION, MQRO_EXPIRATION_WITH_DATA, and MQRO_EXPIRATION_WITH_FULL_DATA.

Confirm-on-arrival options: Specify one of the options listed below to request a confirm-on-arrival report message.

MQRO_COA

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 can be retrieved only if the unit of work is committed.

A COA report is not generated if the *Format* field in the message descriptor is MQFMT_XMIT_Q_HEADER or MQFMT_DEAD_LETTER_HEADER. This prevents a COA report being generated if the message is put on a transmission queue, or is undeliverable and put on a dead-letter queue.

Do not specify more than one of MQRO_COA, MQRO_COA_WITH_DATA, and MQRO_COA_WITH_FULL_DATA.

MQRO_COA_WITH_DATA

This is the same as MQRO_COA, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of MQRO_COA, MQRO_COA_WITH_DATA, and MQRO_COA_WITH_FULL_DATA.

MQRO_COA_WITH_FULL_DATA

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

Do not specify more than one of MQRO_COA, MQRO_COA_WITH_DATA, and MQRO_COA_WITH_FULL_DATA.

Confirm-on-delivery options: Specify one of the options listed below to request a confirm-on-delivery report message.

MQRO_COD

This type of report is generated by the queue manager when an application retrieves the message from the destination queue in a way that deletes the message from the queue. Message data from the original message is not included with the report message.

If the message is retrieved as part of a unit of work, the report message is generated within the same unit of work, so that the report is not available until the unit of work is committed. If the unit of work is backed out, the report is not sent.

A COD report is not always generated if a message is retrieved with the MQGMO_MARK_SKIP_BACKOUT option. If the primary unit of work is backed out but the secondary unit of work is committed, the message is removed from the queue, but a COD report is not generated.

A COD report is not generated if the *Format* field in the message descriptor is MQFMT_DEAD_LETTER_HEADER. This prevents a COD report being generated if the message is undeliverable and put on a dead-letter queue.

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

Do not specify more than one of MQRO_COD, MQRO_COD_WITH_DATA, and MQRO_COD_WITH_FULL_DATA.

MQRO_COD_WITH_DATA

This is the same as MQRO_COD, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

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

- On z/OS, it is the minimum of:
 - The length of the original message
 - The length of the buffer used to retrieve the message
 - 100 bytes.
- In other environments, it is the minimum of:
 - The length of the original message
 - 100 bytes.

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

Do not specify more than one of MQRO_COD, MQRO_COD_WITH_DATA, and MQRO_COD_WITH_FULL_DATA.

MQRO_COD_WITH_FULL_DATA

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

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

Do not specify more than one of MQRO_COD, MQRO_COD_WITH_DATA, and MQRO_COD_WITH_FULL_DATA.

Action-notification options: Specify one or both of the options listed below to request that the receiving application send a positive-action or negative-action report message.

MQRO_PAN

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 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 on this option. The retrieving application must generate the report if appropriate.

MQRO_NAN

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 any data is to be included with the report. For example, you might want 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 on this option. The retrieving application must generate the report if appropriate.

The application must determine which conditions correspond to a positive action and which correspond to a negative action. However, if the request has been only partially performed, generate a NAN report rather than a PAN report if requested. Every possible condition must correspond to either a positive action, or a negative action, but not both.

Message-identifier options: Specify one of the options listed below to control how the *MsgId* of the report message (or of the reply message) is to be set.

MQRO_NEW_MSG_ID

This is the default action, and indicates that if a report or reply is generated as a result of this message, a new *MsgId* is generated for the report or reply message.

MQRO_PASS_MSG_ID

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

The *MsgId* of a publication message will be different for each subscriber that receives a copy of the publication and therefore the *MsgId* copied into the report or reply message will be different for each one.

If this option is not specified, MQRO_NEW_MSG_ID is assumed.

Correlation-identifier options: Specify one of the options listed below to control how the *CorrelId* of the report message (or of the reply message) is to be set.

MQRO_COPY_MSG_ID_TO_CORREL_ID

This is the default action, and indicates that if a report or reply is generated as a result of this message, the *MsgId* of this message is copied to the *CorrelId* of the report or reply message.

The *MsgId* of a publication message will be different for each subscriber that receives a copy of the publication and therefore the *MsgId* copied into the *CorrelId* of the report or reply message will be different for each one.

MQRO_PASS_CORREL_ID

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

The *CorrelId* of a publication message will be specific to a subscriber unless it uses the MQSO_SET_CORREL_ID option and sets the SubCorrelId field in the MQSD to MQCI_NONE. Therefore it is possible that the *CorrelId* copied into the *CorrelId* of the report or reply message will be different for each one.

If this option is not specified, MQRO_COPY_MSG_ID_TO_CORREL_ID is assumed.

Servers replying to requests or generating report messages must check whether the MQRO_PASS_MSG_ID or MQRO_PASS_CORREL_ID options were set in the original message. If they were, the servers must take the action described for those options. If neither is set, the servers must take the corresponding default action.

Disposition options: Specify one of the options listed below to control the disposition of the original message when it cannot be delivered to the destination queue. The application can set the disposition options independently of requesting exception reports.

MQRO_DEAD_LETTER_Q

This is the default action, and places the message on the dead-letter queue if the message cannot be delivered to the destination queue. This happens in the following situations:

- When the application that put the original message cannot be notified synchronously of the problem by means of the reason code returned by the MQPUT or MQPUT1 call. An exception report message is generated, if one was requested by the sender.
- When the application that put the original message was putting to a topic

An exception report message is generated, if one was requested by the sender.

MQRO_DISCARD_MSG

This discards the message if it cannot be delivered to the destination queue. This happens in the following situations:

- When the application that put the original message cannot be notified synchronously of the problem by means of the reason code returned by the MQPUT or MQPUT1 call. An exception report message is generated, if one was requested by the sender.
- When the application that put the original message was putting to a topic

An exception report message is generated, if one was requested by the sender.

If you want to return the original message to the sender, without the original message being placed on the dead-letter queue, the sender must specify MQRO_DISCARD_MSG with MQRO_EXCEPTION_WITH_FULL_DATA.

MQRO_PASS_DISCARD_AND_EXPIRY

If this option is set on a message, and a report or reply is generated because of it, the message descriptor of the report inherits:

- MQRO_DISCARD_MSG if it was set.
- The remaining expiry time of the message (if this is not an expiry report). If this is an expiry report the expiry time is set to 60 seconds.

Activity option

MQRO_ACTIVITY

Using this value allows the route of **any** message to be traced throughout a queue manager network. The report option can be specified on any current user message, instantly allowing you to begin calculating the route of the message through the network.

If the application generating the message cannot switch on activity reports, reports can be turned on using an API crossing exit supplied by queue manager administrators.

Note:

1. The fewer the queue managers in the network that are able to generate activity reports, the less detailed the route.
2. The activity reports might be difficult to place in the correct order to determine the route taken.
3. The activity reports might not be able to find a route to their requested destination.
4. Messages with this report option set must be accepted by any queue manager, even if they do not understand the option. This allows the report option to be set on any user message, even if they are processed by a non Version 6.0 or later queue manager.
5. If a process, either a queue manager or a user process, performs an activity on a message with this option set it can choose to generate and put an activity report.

Default option: Specify the following if no report options are required:

MQRO_NONE

Use this value to indicate that no other options have been specified.

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

General information:

1. All report types required must be specifically requested by the application sending the original message. For example, if a COA report is requested but an exception report is not, a COA report is generated when the message is placed on the destination queue, but no exception report is generated if the destination queue is full when the message arrives there. If no *Report* options are set, no report messages are generated by the queue manager or message channel agent (MCA).

Some report options can be specified even though the local queue manager does not recognize them; this is useful when the option is to be processed by the *destination* queue manager. See Chapter 7, "Report options and message flags," on page 671 for more details.

If a report message is requested, the name of the queue to which to send the report must be specified in the *ReplyToQ* field. When a report message is received, the nature of the report can be determined by examining the *Feedback* field in the message descriptor.

2. If the queue manager or MCA that generates a report message cannot 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 that generated the exception report is left on its transmission queue; this ensures that the message is not lost.
- For all other report types, the report message is discarded and processing continues normally. This is done because either the original message has already been delivered safely (for COA or COD report messages), or is no longer of any interest (for an expiration report message).

Once a report message has been placed successfully on a queue (either the destination queue or an intermediate transmission queue), the message is no longer subject to special processing; it is treated just like any other message.

3. When the report is generated, the *ReplyToQ* queue is opened and the report message put using the authority of the *UserIdentifier* in the MQMD of the message causing the report, except in the following cases:
 - Exception reports generated by a receiving MCA are put with whatever authority the MCA used when it tried to put the message causing the report. The *PutAuthority* channel attribute determines the user identifier used.
 - COA reports generated by the queue manager are put with whatever authority was used when the message causing the report was put on the queue manager generating the report. For example, if the message was put by a receiving MCA using the MCA's user identifier, the queue manager puts the COA report using the MCA's user identifier.

Applications generating reports must use the same authority as they use to generate a reply; this is usually the authority of the user identifier in the original message.

If the report has to travel to a remote destination, senders and receivers can decide whether to accept it, in the same way as they do for other messages.

4. If a report message with data is requested:
 - The report message is always generated with the amount of data requested by the sender of the original message. If the report message is too big for the reply queue, the processing described above occurs; the report message is never truncated to fit on the reply queue.
 - If the *Format* of the original message is MQFMT_XMIT_Q_HEADER, the data included in the report does not include the MQXQH. The report data starts with the first byte of the data beyond the MQXQH in the original message. This occurs whether or not the queue is a transmission queue.
5. If a COA, COD, or expiration report message is received at the reply queue, it is guaranteed that the original message arrived, was delivered, or expired, as appropriate. However, if one or more of these report messages is requested and is *not* received, the reverse cannot be assumed, because one of the following might have occurred:
 - a. The report message is held up because a link is down.
 - b. The report message is held up because a blocking condition exists at an intermediate transmission queue or at the reply queue (for example, the queue is full or inhibited for puts).
 - c. The report message is on a dead-letter queue.
 - d. When the queue manager was attempting to generate the report message, it could neither put it on the appropriate queue, nor on the dead-letter queue, so the report message could not be generated.

- e. A failure of the queue manager occurred between the action being reported (arrival, delivery, or expiry), and generation of the corresponding report message. (This does not happen for COD report messages if the application retrieves the original message within a unit of work, as the COD report message is generated within the same unit of work.)

Exception report messages can be held up in the same way for reasons 1, 2, and 3 above. However, when an MCA cannot generate an exception report message (the report message cannot be put either on the reply queue or the dead-letter queue), the original message remains on the transmission queue at the sender, and the channel is closed. This occurs irrespective of whether the report message was to be generated at the sending or the receiving end of the channel.

6. If the original message is temporarily blocked (resulting in an exception report message being generated and the original message being put on a dead-letter queue), but the blockage clears and an application then reads the original message from the dead-letter queue and puts it again to its destination, the following might 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, because the original message might encounter another blockage later.

Report messages when putting to a topic:

1. Reports can be generated when putting a message to a topic. This message will be sent to all subscribers to the topic, which could be zero, one or many. This should be taken into account when choosing to use report options as many report messages could be generated as a result.
2. When putting a message to a topic, there may be many destination queues that are to be given a copy of the message. If some of these destination queues have a problem, such as queue full, then the successful completion of the MQPUT depends on the setting of NPMSGDLV or PMSGDLV (depending on the persistence of the message). If the setting is such that message delivery to the destination queue must be successful (for example, it is a persistent message to a durable subscriber and PMSGDLV is set to ALL or ALLDUR), then success is defined as one of the following criteria being met:
 - Successful put to the subscriber queue
 - Use of MQRO_DEAD_LETTER_Q and a successful put to the Dead-letter queue if the subscriber queue cannot take the message
 - Use of MQRO_DISCARD_MSG if the subscriber queue cannot take the message.

Report messages for message segments:

1. Report messages can be requested for messages that have segmentation allowed (see the description of the MQMF_SEGMENTATION_ALLOWED flag). If the queue manager finds it necessary to segment the message, a report message can be generated for each of the segments that subsequently encounters the relevant condition. Applications must be prepared to receive multiple report messages for each type of report message requested. Use the *GroupId* field in the report message to correlate the multiple reports with the group identifier of the original message, and the *Feedback* field identify the type of each report message.

2. If MQGMO_LOGICAL_ORDER is used to retrieve report messages for segments, be aware that reports of *different types* might 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 might return the COA and COD report messages interleaved in an unpredictable fashion. Avoid this by using the MQGMO_COMPLETE_MSG option (optionally with MQGMO_ACCEPT_TRUNCATED_MSG). MQGMO_COMPLETE_MSG causes the queue manager to reassemble report messages that have the same report type. For example, the first MQGET call might reassemble all the COA messages relating to the original message, and the second MQGET call might reassemble all the COD messages. Which is reassembled first depends on which type of report message occurs first on the queue.
3. Applications that themselves put segments can specify different report options for each segment. However, note the following points:
 - If the segments are retrieved using the MQGMO_COMPLETE_MSG option, only the report options in the *first* segment are honored by the queue manager.
 - If the segments are retrieved one by one, and most of them have one of the MQRO_COD_* options, but at least one segment does not, you cannot use the MQGMO_COMPLETE_MSG option to retrieve the report messages with a single MQGET call, or use the MQGMO_ALL_SEGMENTS_AVAILABLE option to detect when all the report messages have arrived.
4. In an MQ network, the queue managers can have different 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 does not by default include the necessary segment information in the report message, and this might make it difficult to identify the original message that caused the report to be generated. Avoid this difficulty by requesting data with the report message, that is, by specifying the appropriate MQRO*_WITH_DATA or MQRO*_WITH_FULL_DATA options. However, be aware that if MQRO*_WITH_DATA is specified, *less than* 100 bytes of application message data might be returned to the application that retrieves the report message, if the report message is generated by a queue manager or MCA that does not support segmentation.

Contents of the message descriptor for a report message: When the queue manager or message channel agent (MCA) generates a report message, it sets the fields in the message descriptor to the following values, and then puts the message in the normal way.

Field in MQMD	Value used
<i>StrucId</i>	MQMD_STRUC_ID
<i>Version</i>	MQMD_VERSION_2
<i>Report</i>	MQRO_NONE
<i>MsgType</i>	MQMT_REPORT
<i>Expiry</i>	MQEI_UNLIMITED
<i>Feedback</i>	As appropriate for the nature of the report (MQFB_COA, MQFB_COD, MQFB_EXPIRATION, or an MQRC_* value)
<i>Encoding</i>	Copied from the original message descriptor
<i>CodedCharSetId</i>	Copied from the original message descriptor
<i>Format</i>	Copied from the original message descriptor
<i>Priority</i>	Copied from the original message descriptor
<i>Persistence</i>	Copied from the original message descriptor

Field in MQMD	Value used
<i>MsgId</i>	As specified by the report options in the original message descriptor
<i>CorrelId</i>	As specified by the report options in the original message descriptor
<i>BackoutCount</i>	0
<i>ReplyToQ</i>	Blanks
<i>ReplyToQMGr</i>	Name of queue manager
<i>UserIdentifier</i>	As set by the MQPMO_PASS_IDENTITY_CONTEXT option
<i>AccountingToken</i>	As set by the MQPMO_PASS_IDENTITY_CONTEXT option
<i>ApplIdentityData</i>	As set by the MQPMO_PASS_IDENTITY_CONTEXT option
<i>PutApplType</i>	MQAT_QMGR, or as appropriate for the message channel agent
<i>PutApplName</i>	First 28 bytes of the queue-manager name or message channel agent name. For report messages generated by the IMS bridge, this field contains the XCF group name and XCF member name of the IMS system to which the message relates.
<i>PutDate</i>	Date when report message is sent
<i>PutTime</i>	Time when report message is sent
<i>ApplOriginData</i>	Blanks
<i>GroupId</i>	Copied from the original message descriptor
<i>MsgSeqNumber</i>	Copied from the original message descriptor
<i>Offset</i>	Copied from the original message descriptor
<i>MsgFlags</i>	Copied from the original message descriptor
<i>OriginalLength</i>	Copied from the original message descriptor if not MQOL_UNDEFINED, and set to the length of the original message data otherwise

An application generating a report is recommended to set similar values, except for the following:

- The *ReplyToQMGr* field can be set to blanks (the queue manager changes this to the name of the local queue manager when the message is put).
- Set the context fields using the option that would have been used for a reply, normally MQPMO_PASS_IDENTITY_CONTEXT.

Analyzing the report field: The *Report* field contains subfields; because of this, applications that need to check whether the sender of the message requested a particular report must use one of the techniques described in “Analyzing the report field” on page 673.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQRO_NONE.

StruclD (MQCHAR4)

This is the structure identifier, and must be:

MQMD_STRUC_ID

Identifier for message descriptor structure.

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

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

UserIdentifier (MQCHAR12)

This is part of the **identity context** of the message. For more information about message context, see “Overview for MQMD” on page 182; also see the *WebSphere MQ Application Programming Guide*.

UserIdentifier specifies the user identifier of the application that originated the message. The queue manager treats this information as character data, but does not define the format of it.

After a message has been received, use *UserIdentifier* in the *AlternateUserId* field of the *ObjDesc* parameter of a subsequent MQOPEN or MQPUT1 call to perform the authorization check for the *UserIdentifier* user instead of the application performing the open.

When the queue manager generates this information for an MQPUT or MQPUT1 call:

- On z/OS, the queue manager uses the *AlternateUserId* from the *ObjDesc* parameter of the MQOPEN or MQPUT1 call if the MQOO_ALTERNATE_USER_AUTHORITY or MQPMO_ALTERNATE_USER_AUTHORITY option was specified. If the relevant option was not specified, the queue manager uses a user identifier determined from the environment.
- In other environments, the queue manager always uses a user identifier determined from the environment.

When the user identifier is determined from the environment:

- On z/OS, the queue manager uses:
 - For MVS (batch), the user identifier from the JES JOB card or started task
 - For TSO, the user identifier propagated to the job during job submission
 - For CICS, the user identifier associated with the task
 - For IMS, the user identifier depends on the type of application:
 - For:
 - Nonmessage BMP regions
 - Nonmessage IFP regions
 - Message BMP and message IFP regions that have *not* issued a successful GU call

the queue manager uses the user identifier from the region JES JOB card or the TSO user identifier. If these are blank or null, it uses the name of the program specification block (PSB).

- For:
 - Message BMP and message IFP regions that *have* issued a successful GU call
 - MPP regions

the queue manager uses one of:

- The signed-on user identifier associated with the message
- The logical terminal (LTERM) name
- The user identifier from the region JES JOB card
- The TSO user identifier
- The PSB name

- On i5/OS, the queue manager uses the name of the user profile associated with the application job.
- On UNIX systems, the queue manager uses:
 - The application's logon name
 - The effective user identifier of the process if no logon is available
 - The user identifier associated with the transaction, if the application is a CICS transaction
- On Windows systems, the queue manager uses the first 12 characters of the logged-on user name.

This field is normally an output field generated by the queue manager but for an MQPUT or MQPUT1 call you can make this field an input/output field and specify the *UserIdentifier* field instead of letting the queue manager generate this information. Specify either MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT in the *PutMsgOpts* parameter and specify a *userid* in the *UserIdentifier* field if you do not want the queue manager to generate the *UserIdentifier* field for an MQPUT or MQPUT1 call.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. Any information following a null character within the field is discarded. The queue manager converts the null character and any following characters to blanks. If MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *UserIdentifier* that was transmitted with the message if it was put to a queue. This will be the value of *UserIdentifier* that is kept with the message if it is retained (see description of MQPMO_RETAIN for more details about retained publications) but is not used as the *UserIdentifier* when the message is sent as a publication to subscribers since they provide a value to override *UserIdentifier* in all publications sent to them. If the message has no context, the field is entirely blank.

This is an output field for the MQGET call. The length of this field is given by MQ_USER_ID_LENGTH. The initial value of this field is the null string in C, and 12 blank characters in other programming languages.

Version (MQLONG)

This is the structure version number, and must be one of the following:

MQMD_VERSION_1

Version-1 message descriptor structure.

This version is supported in all environments.

MQMD_VERSION_2

Version-2 message descriptor structure.

This version is supported in all WebSphere MQ V6.0 and later environments, plus WebSphere MQ clients connected to these systems.

Note: When a version-2 MQMD is used, the queue manager performs additional checks on any MQ header structures that might be present at the beginning of the application message data; for further details see the usage notes for the MQPUT call.

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

MQMD_CURRENT_VERSION

Current version of message descriptor structure.

This is always an input field. The initial value of this field is MQMD_VERSION_1.

Initial values and language declarations for MQMD

Table 47. Initial values of fields in MQMD for MQMD

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQMD_STRUC_ID	'Mdbbb'
<i>Version</i>	MQMD_VERSION_1	1
<i>Report</i>	MQRO_NONE	0
<i>MsgType</i>	MQMT_DATAGRAM	8
<i>Expiry</i>	MQEI_UNLIMITED	-1
<i>Feedback</i>	MQFB_NONE	0
<i>Encoding</i>	MQENC_NATIVE	Depends on environment
<i>CodedCharSetId</i>	MQCCSI_Q_MGR	0
<i>Format</i>	MQFMT_NONE	Blanks
<i>Priority</i>	MQPRI_PRIORITY_AS_Q_DEF	-1
<i>Persistence</i>	MQPER_PERSISTENCE_AS_Q_DEF	2
<i>MsgId</i>	MQMI_NONE	Nulls
<i>CorrelId</i>	MQCI_NONE	Nulls
<i>BackoutCount</i>	None	0
<i>ReplyToQ</i>	None	Null string or blanks
<i>ReplyToQMgr</i>	None	Null string or blanks
<i>UserIdentifier</i>	None	Null string or blanks
<i>AccountingToken</i>	MQACT_NONE	Nulls
<i>ApplIdentityData</i>	None	Null string or blanks
<i>PutApplType</i>	MQAT_NO_CONTEXT	0
<i>PutApplName</i>	None	Null string or blanks
<i>PutDate</i>	None	Null string or blanks
<i>PutTime</i>	None	Null string or blanks
<i>ApplOriginData</i>	None	Null string or blanks
<i>GroupId</i>	MQGI_NONE	Nulls
<i>MsgSeqNumber</i>	None	1
<i>Offset</i>	None	0
<i>MsgFlags</i>	MQMF_NONE	0

Table 47. Initial values of fields in MQMD for MQMD (continued)

Field name	Name of constant	Value of constant
<i>OriginalLength</i>	MQOL_UNDEFINED	-1
Notes:		
1. The symbol <i>b</i> represents a single blank character.		
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.		
3. In the C programming language, the macro variable MQMD_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:		
MQMD MyMD = {MQMD_DEFAULT};		

C declaration

```
typedef struct tagMQMD MQMD;
struct tagMQMD {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQLONG    Report;           /* Options for report messages */
    MQLONG    MsgType;          /* Message type */
    MQLONG    Expiry;           /* Message lifetime */
    MQLONG    Feedback;         /* Feedback or reason code */
    MQLONG    Encoding;         /* Numeric encoding of message data */
    MQLONG    CodedCharSetId;   /* Character set identifier of message
                                data */
    MQCHAR8   Format;           /* Format name of message data */
    MQLONG    Priority;          /* Message priority */
    MQLONG    Persistence;      /* Message persistence */
    MQBYTE24  MsgId;            /* Message identifier */
    MQBYTE24  CorrelId;         /* Correlation identifier */
    MQLONG    BackoutCount;     /* Backout counter */
    MQCHAR48  ReplyToQ;         /* Name of reply queue */
    MQCHAR48  ReplyToQMGr;      /* Name of reply queue manager */
    MQCHAR12  UserIdentifier;    /* User identifier */
    MQBYTE32  AccountingToken;  /* Accounting token */
    MQCHAR32  ApplIdentityData; /* Application data relating to
                                identity */
    MQLONG    PutAppIType;      /* Type of application that put the
                                message */
    MQCHAR28  PutAppIName;      /* Name of application that put the
                                message */
    MQCHAR8   PutDate;          /* Date when message was put */
    MQCHAR8   PutTime;          /* Time when message was put */
    MQCHAR4   ApplOriginData;   /* Application data relating to origin */
    MQBYTE24  GroupId;          /* Group identifier */
    MQLONG    MsgSeqNumber;     /* Sequence number of logical message
                                within group */
    MQLONG    Offset;           /* Offset of data in physical message
                                from start of logical message */
    MQLONG    MsgFlags;         /* Message flags */
    MQLONG    OriginalLength;   /* Length of original message */
};
```

COBOL declaration

```
** MQMD structure
10 MQMD.
** Structure identifier
15 MQMD-STRUCID PIC X(4).
** Structure version number
15 MQMD-VERSION PIC S9(9) BINARY.
```

```

** Options for report messages
  15 MQMD-REPORT          PIC S9(9) BINARY.
** Message type
  15 MQMD-MSGTYPE        PIC S9(9) BINARY.
** Message lifetime
  15 MQMD-EXPIRY         PIC S9(9) BINARY.
** Feedback or reason code
  15 MQMD-FEEDBACK       PIC S9(9) BINARY.
** Numeric encoding of message data
  15 MQMD-ENCODING       PIC S9(9) BINARY.
** Character set identifier of message data
  15 MQMD-CODEDCHARSETID PIC S9(9) BINARY.
** Format name of message data
  15 MQMD-FORMAT         PIC X(8).
** Message priority
  15 MQMD-PRIORITY       PIC S9(9) BINARY.
** Message persistence
  15 MQMD-PERSISTENCE    PIC S9(9) BINARY.
** Message identifier
  15 MQMD-MSGID          PIC X(24).
** Correlation identifier
  15 MQMD-CORRELID       PIC X(24).
** Backout counter
  15 MQMD-BACKOUTCOUNT  PIC S9(9) BINARY.
** Name of reply queue
  15 MQMD-REPLYTOQ       PIC X(48).
** Name of reply queue manager
  15 MQMD-REPLYTOQMGR    PIC X(48).
** User identifier
  15 MQMD-USERIDENTIFIER PIC X(12).
** Accounting token
  15 MQMD-ACCOUNTINGTOKEN PIC X(32).
** Application data relating to identity
  15 MQMD-APPLIDENTITYDATA PIC X(32).
** Type of application that put the message
  15 MQMD-PUTAPPLTYPE    PIC S9(9) BINARY.
** Name of application that put the message
  15 MQMD-PUTAPPLNAME    PIC X(28).
** Date when message was put
  15 MQMD-PUTDATE        PIC X(8).
** Time when message was put
  15 MQMD-PUTTIME        PIC X(8).
** Application data relating to origin
  15 MQMD-APPLORIGINDATA PIC X(4).
** Group identifier
  15 MQMD-GROUPID        PIC X(24).
** Sequence number of logical message within group
  15 MQMD-MSGSEQNUMBER   PIC S9(9) BINARY.
** Offset of data in physical message from start of logical message
  15 MQMD-OFFSET         PIC S9(9) BINARY.
** Message flags
  15 MQMD-MSGFLAGS       PIC S9(9) BINARY.
** Length of original message
  15 MQMD-ORIGINALLENGTH PIC S9(9) BINARY.

```

PL/I declaration

```

dcl
  1 MQMD based,
    3 StrucId          char(4),          /* Structure identifier */
    3 Version          fixed bin(31),    /* Structure version number */
    3 Report           fixed bin(31),    /* Options for report messages */
    3 MsgType          fixed bin(31),    /* Message type */
    3 Expiry           fixed bin(31),    /* Message lifetime */
    3 Feedback         fixed bin(31),    /* Feedback or reason code */
    3 Encoding         fixed bin(31),    /* Numeric encoding of message
                                         data */
    3 CodedCharSetId  fixed bin(31),    /* Character set identifier of

```

```

message data */
3 Format          char(8),      /* Format name of message data */
3 Priority        fixed bin(31), /* Message priority */
3 Persistence    fixed bin(31), /* Message persistence */
3 MsgId          char(24),     /* Message identifier */
3 CorrelId       char(24),     /* Correlation identifier */
3 BackoutCount   fixed bin(31), /* Backout counter */
3 ReplyToQ       char(48),     /* Name of reply queue */
3 ReplyToQMGR    char(48),     /* Name of reply queue manager */
3 UserIdentifier char(12),     /* User identifier */
3 AccountingToken char(32),    /* Accounting token */
3 ApplIdentityData char(32),   /* Application data relating to
identity */
3 PutApplType    fixed bin(31), /* Type of application that put the
message */
3 PutApplName    char(28),     /* Name of application that put the
message */
3 PutDate        char(8),      /* Date when message was put */
3 PutTime        char(8),      /* Time when message was put */
3 ApplOriginData char(4),      /* Application data relating to
origin */
3 GroupId        char(24),     /* Group identifier */
3 MsgSeqNumber   fixed bin(31), /* Sequence number of logical
message within group */
3 Offset         fixed bin(31), /* Offset of data in physical
message from start of logical
message */
3 MsgFlags       fixed bin(31), /* Message flags */
3 OriginalLength fixed bin(31); /* Length of original message */

```

System/390 assembler declaration

```

MQMD             DSECT
MQMD_STRUCID     DS   CL4   Structure identifier
MQMD_VERSION     DS   F     Structure version number
MQMD_REPORT      DS   F     Options for report messages
MQMD_MSGTYPE     DS   F     Message type
MQMD_EXPIRY      DS   F     Message lifetime
MQMD_FEEDBACK    DS   F     Feedback or reason code
MQMD_ENCODING    DS   F     Numeric encoding of message data
MQMD_CODEDCHARSETID DS   F   Character set identifier of message
*               data
MQMD_FORMAT      DS   CL8   Format name of message data
MQMD_PRIORITY    DS   F     Message priority
MQMD_PERSISTENCE DS   F     Message persistence
MQMD_MSGID       DS   XL24  Message identifier
MQMD_CORRELID    DS   XL24  Correlation identifier
MQMD_BACKOUTCOUNT DS   F   Backout counter
MQMD_REPLYTOQ    DS   CL48  Name of reply queue
MQMD_REPLYTOQMGR DS   CL48  Name of reply queue manager
MQMD_USERIDENTIFIER DS   CL12 User identifier
MQMD_ACCOUNTINGTOKEN DS   XL32 Accounting token
MQMD_APPLIDENTITYDATA DS   CL32 Application data relating to identity
MQMD_PUTAPPLTYPE DS   F     Type of application that put the
*               message
MQMD_PUTAPPLNAME DS   CL28  Name of application that put the
*               message
MQMD_PUTDATE     DS   CL8   Date when message was put
MQMD_PUTTIME     DS   CL8   Time when message was put
MQMD_APPLORIGINDATA DS   CL4  Application data relating to origin
MQMD_GROUPID     DS   XL24  Group identifier
MQMD_MSGSEQNUMBER DS   F     Sequence number of logical message
*               within group
MQMD_OFFSET      DS   F     Offset of data in physical message
*               from start of logical message
MQMD_MSGFLAGS    DS   F     Message flags
MQMD_ORIGINALLENGTH DS   F   Length of original message
*

```

```

MQMD_LENGTH      EQU  *-MQMD
                  ORG  MQMD
MQMD_AREA        DS   CL(MQMD_LENGTH)

```

Visual Basic declaration

```

Type MQMD
  StrucId      As String*4  'Structure identifier'
  Version      As Long      'Structure version number'
  Report       As Long      'Options for report messages'
  MsgType      As Long      'Message type'
  Expiry       As Long      'Message lifetime'
  Feedback     As Long      'Feedback or reason code'
  Encoding     As Long      'Numeric encoding of message data'
  CodedCharSetId As Long    'Character set identifier of message'
                  'data'

  Format       As String*8  'Format name of message data'
  Priority     As Long      'Message priority'
  Persistence  As Long      'Message persistence'
  MsgId       As MBYTE24   'Message identifier'
  CorrelId    As MBYTE24   'Correlation identifier'
  BackoutCount As Long      'Backout counter'
  ReplyToQ    As String*48 'Name of reply queue'
  ReplyToQMgr As String*48 'Name of reply queue manager'
  UserIdentifier As String*12 'User identifier'
  AccountingToken As MBYTE32 'Accounting token'
  ApplIdentityData As String*32 'Application data relating to identity'
  PutApplType  As Long      'Type of application that put the'
                  'message'

  PutApplName  As String*28 'Name of application that put the'
                  'message'

  PutDate      As String*8  'Date when message was put'
  PutTime      As String*8  'Time when message was put'
  ApplOriginData As String*4  'Application data relating to origin'
  GroupId      As MBYTE24   'Group identifier'
  MsgSeqNumber As Long      'Sequence number of logical message'
                  'within group'

  Offset       As Long      'Offset of data in physical message'
                  'from start of logical message'

  MsgFlags     As Long      'Message flags'
  OriginalLength As Long    'Length of original message'
End Type

```

MQMDE – Message descriptor extension

The following table summarizes the fields in the structure.

Table 48. Fields in MQMDE

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>StrucLength</i>	Length of MQMDE structure	StrucLength
<i>Encoding</i>	Numeric encoding of data that follows MQMDE	Encoding
<i>CodedCharSetId</i>	Character set identifier of data that follows MQMDE	CodedCharSetId
<i>Format</i>	Format name of data that follows MQMDE	Format
<i>Flags</i>	General flags	Flags
<i>GroupId</i>	Group identifier	GroupId

Table 48. Fields in MQMDE (continued)

Field	Description	Topic
<i>MsgSeqNumber</i>	Sequence number of logical message within group	MsgSeqNumber
<i>Offset</i>	Offset of data in physical message from start of logical message	Offset
<i>MsgFlags</i>	Message flags	MsgFlags
<i>OriginalLength</i>	Length of original message	OriginalLength

Overview for MQMDE

Availability: All WebSphere MQ systems, plus WebSphere MQ clients connected to these systems.

Purpose: The MQMDE structure describes the data that sometimes occurs preceding the application message data. The structure contains those MQMD fields that exist in the version-2 MQMD, but not in the version-1 MQMD.

Format name: MQFMT_MD_EXTENSION.

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

Set the character set and encoding of the MQMDE into the *CodedCharSetId* and *Encoding* fields in:

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

If the MQMDE is not in the queue manager's character set and encoding, the MQMDE is accepted but not honored, that is, the MQMDE is treated as message data.

Note: On Windows, applications compiled with Micro Focus COBOL use a value of MQENC_NATIVE that is different from the queue-manager's encoding. Although numeric fields in the MQMD structure on the MQPUT, MQPUT1, and MQGET calls must be in the Micro Focus COBOL encoding, numeric fields in the MQMDE structure must be in the queue-manager's encoding. This latter is given by MQENC_NATIVE for the C programming language, and has the value 546.

Usage: Applications that use a version-2 MQMD will not encounter an MQMDE structure. However, specialized applications, and applications that continue to use a version-1 MQMD, might encounter an MQMDE in some situations. The MQMDE structure can occur in the following circumstances:

- Specified on the MQPUT and MQPUT1 calls
- Returned by the MQGET call
- In messages on transmission queues

These are described below.

MQMDE specified on MQPUT and MQPUT1 calls: On the MQPUT and MQPUT1 calls, if the application provides a version-1 MQMD, the application can optionally prefix the message data with an MQMDE, setting the *Format* field in

MQMD to MQFMT_MD_EXTENSION to indicate that an MQMDE is present. If the application does not provide an MQMDE, the queue manager assumes default values for the fields in the MQMDE. The default values that the queue manager uses are the same as the initial values for the structure; see Table 50 on page 244.

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 49.

Table 49. Queue-manager action when MQMDE specified on MQPUT or MQPUT1 for MQMDE

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

Note: On z/OS, if the application specifies a version-1 MQMD with an MQMDE, the queue manager validates the MQMDE only if the queue has an *IndexType* of MQIT_GROUP_ID.

There is one special case. If the application uses a version-2 MQMD to put a message that is a segment (that is, the MQMF_SEGMENT or MQMF_LAST_SEGMENT flag is set), and the format name in the MQMD is MQFMT_DEAD_LETTER_HEADER, the queue manager generates an MQMDE structure and inserts it *between* the MQDLH structure and the data that follows it. In the MQMD that the queue manager retains with the message, the version-2 fields are set to their default values.

Several of the fields that exist in the version-2 MQMD but not the version-1 MQMD are input/output fields on MQPUT and MQPUT1. However, the queue manager does *not* return any values in the equivalent fields in the MQMDE on output from the MQPUT and MQPUT1 calls; if the application requires those output values, it must use a version-2 MQMD.

MQMDE returned by MQGET call: On the MQGET call, if the application provides a version-1 MQMD, the queue manager prefixes the message returned with an MQMDE, but only if one or more of the fields in the MQMDE has a nondefault value. The queue manager sets the *Format* field in MQMD to the value MQFMT_MD_EXTENSION to indicate that an MQMDE is present.

If the application provides an MQMDE at the start of the *Buffer* parameter, the MQMDE is ignored. On return from the MQGET call, it is replaced by the MQMDE for the message (if one is needed), or overwritten by the application message data (if the MQMDE is not needed).

If the MQGET call returns an MQMDE, the data in the MQMDE is usually in the queue manager's character set and encoding. However the MQMDE might be in some other character set and encoding if:

- The MQMDE was treated as data on the MQPUT or MQPUT1 call (see Table 49 on page 241 for the circumstances that can cause this).
- The message was received from a remote queue manager connected by a TCP connection, and the receiving message channel agent (MCA) was not set up correctly (see the *WebSphere MQ Intercommunications* manual for further information).

Note: On Windows, applications compiled with Micro Focus COBOL use a value of MQENC_NATIVE that is different from the queue-manager's encoding (see above).

MQMDE in messages on transmission queues: Messages on transmission queues are prefixed with the MQXQH structure, which contains within it a version-1 MQMD. An MQMDE might also be present, positioned between the MQXQH structure and application message data, but it is usually 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 for MQMDE

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

CodedCharSetId (MQLONG)

This specifies the character set identifier of the data that follows the MQMDE structure; it does not apply to character data in the MQMDE structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that this field is valid. The following special value can be used:

MQCCSI_INHERIT

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

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

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

This value is supported in the following environments: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG)

This specifies the numeric encoding of the data that follows the MQMDE structure; it does not apply to numeric data in the MQMDE structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that the field is valid. See the *Encoding* field described in “MQMD – Message descriptor” on page 181 for more information about data encodings.

The initial value of this field is MQENC_NATIVE.

Flags (MQLONG)

The following flag can be specified:

MQMDEF_NONE
No flags.

The initial value of this field is MQMDEF_NONE.

Format (MQCHAR8)

This specifies the format name of the data that follows the MQMDE structure.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that this field is valid. See the *Format* field described in “MQMD – Message descriptor” on page 181 for more information about format names.

The initial value of this field is MQFMT_NONE.

GroupId (MQBYTE24)

See the *GroupId* field described in “MQMD – Message descriptor” on page 181. The initial value of this field is MQGI_NONE.

MsgFlags (MQLONG)

See the *MsgFlags* field described in “MQMD – Message descriptor” on page 181. The initial value of this field is MQMF_NONE.

MsgSeqNumber (MQLONG)

See the *MsgSeqNumber* field described in “MQMD – Message descriptor” on page 181. The initial value of this field is 1.

Offset (MQLONG)

See the *Offset* field described in “MQMD – Message descriptor” on page 181. The initial value of this field is 0.

OriginalLength (MQLONG)

See the *OriginalLength* field described in “MQMD – Message descriptor” on page 181. The initial value of this field is MQOL_UNDEFINED.

StrucId (MQCHAR4)

The value must be:

MQMDE_STRUC_ID

Identifier for message descriptor extension structure.

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

The initial value of this field is MQMDE_STRUC_ID.

StrucLength (MQLONG)

This is the length of the MQMDE structure; the following value is defined:

MQMDE_LENGTH_2

Length of version-2 message descriptor extension structure.

The initial value of this field is MQMDE_LENGTH_2.

Version (MQLONG)

This is the structure version number; the value must be:

MQMDE_VERSION_2

Version-2 message descriptor extension structure.

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

MQMDE_CURRENT_VERSION

Current version of message descriptor extension structure.

The initial value of this field is MQMDE_VERSION_2.

Initial values and language declarations for MQMDE

Table 50. Initial values of fields in MQMDE for MQMDE

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQMDE_STRUC_ID	'MDEb'
<i>Version</i>	MQMDE_VERSION_2	2
<i>StrucLength</i>	MQMDE_LENGTH_2	72
<i>Encoding</i>	MQENC_NATIVE	Depends on environment
<i>CodedCharSetId</i>	MQCCSI_UNDEFINED	0
<i>Format</i>	MQFMT_NONE	Blanks
<i>Flags</i>	MQMDEF_NONE	0
<i>GroupId</i>	MQGI_NONE	Nulls
<i>MsgSeqNumber</i>	None	1
<i>Offset</i>	None	0
<i>MsgFlags</i>	MQMF_NONE	0
<i>OriginalLength</i>	MQOL_UNDEFINED	-1

Table 50. Initial values of fields in MQMDE for MQMDE (continued)

Field name	Name of constant	Value of constant
Notes:		
1. The symbol <code>b</code> represents a single blank character.		
2. In the C programming language, the macro variable <code>MQMDE_DEFAULT</code> contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:		
<pre>MQMDE MyMDE = {MQMDE_DEFAULT};</pre>		

C declaration

```
typedef struct tagMQMDE MQMDE;
struct tagMQMDE {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQLONG    StrucLength;      /* Length of MQMDE structure */
    MQLONG    Encoding;        /* Numeric encoding of data that follows
                               MQMDE */
    MQLONG    CodedCharSetId;   /* Character-set identifier of data that
                               follows MQMDE */
    MQCHAR8   Format;           /* Format name of data that follows
                               MQMDE */
    MQLONG    Flags;            /* General flags */
    MQBYTE24  GroupId;          /* Group identifier */
    MQLONG    MsgSeqNumber;     /* Sequence number of logical message
                               within group */
    MQLONG    Offset;           /* Offset of data in physical message from
                               start of logical message */
    MQLONG    MsgFlags;         /* Message flags */
    MQLONG    OriginalLength;   /* Length of original message */
};
```

COBOL declaration

```
** MQMDE structure
10 MQMDE.
** Structure identifier
15 MQMDE-STRUCID PIC X(4).
** Structure version number
15 MQMDE-VERSION PIC S9(9) BINARY.
** Length of MQMDE structure
15 MQMDE-STRUCLength PIC S9(9) BINARY.
** Numeric encoding of data that follows MQMDE
15 MQMDE-ENCODING PIC S9(9) BINARY.
** Character-set identifier of data that follows MQMDE
15 MQMDE-CODEDCHARSETID PIC S9(9) BINARY.
** Format name of data that follows MQMDE
15 MQMDE-FORMAT PIC X(8).
** General flags
15 MQMDE-FLAGS PIC S9(9) BINARY.
** Group identifier
15 MQMDE-GROUPID PIC X(24).
** Sequence number of logical message within group
15 MQMDE-MSGSEQNUMBER PIC S9(9) BINARY.
** Offset of data in physical message from start of logical message
15 MQMDE-OFFSET PIC S9(9) BINARY.
** Message flags
15 MQMDE-MSGFLAGS PIC S9(9) BINARY.
** Length of original message
15 MQMDE-ORIGINALLENGTH PIC S9(9) BINARY.
```

PL/I declaration

```
dc1
1 MQMDE based,
3 StrucId      char(4),      /* Structure identifier */
3 Version      fixed bin(31), /* Structure version number */
3 StrucLength  fixed bin(31), /* Length of MQMDE structure */
3 Encoding     fixed bin(31), /* Numeric encoding of data that
                             follows MQMDE */
3 CodedCharSetId fixed bin(31), /* Character-set identifier of data
                             that follows MQMDE */
3 Format        char(8),      /* Format name of data that follows
                             MQMDE */
3 Flags        fixed bin(31), /* General flags */
3 GroupId      char(24),     /* Group identifier */
3 MsgSeqNumber fixed bin(31), /* Sequence number of logical message
                             within group */
3 Offset       fixed bin(31), /* Offset of data in physical message
                             from start of logical message */
3 MsgFlags     fixed bin(31), /* Message flags */
3 OriginalLength fixed bin(31); /* Length of original message */
```

System/390 assembler declaration

```
MQMDE          DSECT
MQMDE_STRUCID  DS CL4  Structure identifier
MQMDE_VERSION  DS F    Structure version number
MQMDE_STRUCLNGTH DS F    Length of MQMDE structure
MQMDE_ENCODING DS F    Numeric encoding of data that follows
*              MQMDE
MQMDE_CODEDCHARSETID DS F    Character-set identifier of data that
*              follows MQMDE
MQMDE_FORMAT   DS CL8  Format name of data that follows MQMDE
MQMDE_FLAGS    DS F    General flags
MQMDE_GROUPID  DS XL24 Group identifier
MQMDE_MSGSEQNUMBER DS F    Sequence number of logical message
*              within group
MQMDE_OFFSET   DS F    Offset of data in physical message from
*              start of logical message
MQMDE_MSGFLAGS DS F    Message flags
MQMDE_ORIGINALLENGTH DS F    Length of original message
*
MQMDE_LENGTH   EQU *-MQMDE
               ORG MQMDE
MQMDE_AREA     DS CL(MQMDE_LENGTH)
```

Visual Basic declaration

```
Type MQMDE
  StrucId      As String*4 'Structure identifier'
  Version      As Long      'Structure version number'
  StrucLength  As Long      'Length of MQMDE structure'
  Encoding     As Long      'Numeric encoding of data that follows'
  Encoding     As Long      'MQMDE'
  CodedCharSetId As Long    'Character-set identifier of data that'
  CodedCharSetId As Long    'follows MQMDE'
  Format        As String*8 'Format name of data that follows MQMDE'
  Flags        As Long      'General flags'
  GroupId      As MQBYTE24 'Group identifier'
  MsgSeqNumber As Long      'Sequence number of logical message within'
  MsgSeqNumber As Long      'group'
  Offset       As Long      'Offset of data in physical message from'
  Offset       As Long      'start of logical message'
  MsgFlags     As Long      'Message flags'
  OriginalLength As Long    'Length of original message'
End Type
```

MQMHBO – Message handle to buffer options

The following table summarizes the fields in the structure. MQMHBO structure - message handle to buffer options

Table 51. Fields in MQMHBO

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>Options</i>	Options controlling the action of MQMHBUF	Options

Overview for MQMHBO

Availability: All WebSphere MQ systems and WebSphere MQ clients.

Purpose: The MQMHBO structure allows applications to specify options that control how buffers are produced from message handles. The structure is an input parameter on the MQMHBUF call.

Character set and encoding: Data in MQMHBO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields for MQMHBO

Message handle to buffer options structure - fields

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

Options (MQLONG)

Message handle to buffer options structure - Options field

These options control the action of MQMHBUF.

You must specify the following option:

MQMHBO_PROPERTIES_IN_MQRFH2

When converting properties from a message handle into a buffer, convert them into the MQRFH2 format.

Optionally, you can also specify the following value. If required values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

MQMHBO_DELETE_PROPERTIES

Properties that are added to the buffer are deleted from the message handle. If the call fails no properties are deleted.

This is always an input field. The initial value of this field is MQMHBO_PROPERTIES_IN_MQRFH2.

StrucId (MQCHAR4)

Message handle to buffer options structure - StrucId field

This is the structure identifier. The value must be:

MQMHBO_STRUC_ID

Identifier for message handle to buffer options structure.

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

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

Version (MQLONG)

Message handle to buffer options structure - Version field

This is the structure version number. The value must be:

MQMHBO_VERSION_1

Version number for message handle to buffer options structure.

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

MQMHBO_CURRENT_VERSION

Current version of message handle to buffer options structure.

This is always an input field. The initial value of this field is MQMHBO_VERSION_1.

Initial values and language declarations for MQMHBO

Message handle to buffer structure - Initial values

Table 52. Initial values of fields in MQMHBO

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQMHBO_STRUC_ID	'MHBO'
<i>Version</i>	MQMHBO_VERSION_1	1
<i>Options</i>	MQMHBO_PROPERTIES_IN_MQRFH2	

Notes:

1. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
2. In the C programming language, the macro variable MQMHBO_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

```
MQMHBO MyMHBO = {MQMHBO_DEFAULT};
```

C declaration

Message handle to buffer options structure - C language declaration

```
typedef struct tagMQMHBO MQMHBO;  
struct tagMQMHBO {  
    MCHAR4  StrucId;          /* Structure identifier */  
    MQLONG  Version;         /* Structure version number */  
    MQLONG  Options;        /* Options that control the action of  
                             MQMHBUF */  
};
```

COBOL declaration

Message handle to buffer options structure - COBOL language declaration

```

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

```

PL/I declaration

Message handle to buffer options structure - PL/I language declaration

```

Dcl
1 MQMHBO based,
3 StrucId char(4), /* Structure identifier */
3 Version fixed bin(31), /* Structure version number */
3 Options fixed bin(31), /* Options that control the action
of MQMHBUF */

```

System/390 assembler declaration

Message handle to buffer options structure - Assembler language declaration

```

MQMHBO DSECT
MQMHBO_STRUCID DS CL4 Structure identifier
MQMHBO_VERSION DS F Structure version number
MQMHBO_OPTIONS DS F Options that control the
* action of MQMHBUF
MQMHBO_LENGTH EQU *-MQMHBO
MQMHBO_AREA DS CL(MQMHBO_LENGTH)

```

MQOD – Object descriptor

The following table summarizes the fields in the structure.

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>ObjectType</i>	Object type	ObjectType
<i>ObjectName</i>	Object name	ObjectName
<i>ObjectQMgrName</i>	Object queue manager name	ObjectQMgrName
<i>DynamicQName</i>	Dynamic queue name	DynamicQName
<i>AlternateUserId</i>	Alternate user identifier	AlternateUserId
Note: The remaining fields are ignored if <i>Version</i> is less than MQOD_VERSION_2.		
<i>RecsPresent</i>	Number of object records present	RecsPresent
<i>KnownDestCount</i>	Number of local queues opened successfully	KnownDestCount
<i>UnknownDestCount</i>	Number of remote queues opened successfully	UnknownDestCount
<i>InvalidDestCount</i>	Number of queues that failed to open	InvalidDestCount
<i>ObjectRecOffset</i>	Offset of first object record from start of MQOD	ObjectRecOffset
<i>ResponseRecOffset</i>	Offset of first response record from start of MQOD	ResponseRecOffset
<i>ObjectRecPtr</i>	Address of first object record	ObjectRecPtr
<i>ResponseRecPtr</i>	Address of first response record	ResponseRecPtr

Field	Description	Topic
Note: The remaining fields are ignored if <i>Version</i> is less than MQOD_VERSION_3.		
<i>AlternateSecurityId</i>	Alternate security identifier	AlternateSecurityId
<i>ResolvedQName</i>	Resolved queue name	ResolvedQName
<i>ResolvedQMgrName</i>	Resolved queue manager name	ResolvedQMgrName
Note: The remaining fields are ignored if <i>Version</i> is less than MQOD_VERSION_4.		
<i>ObjectString</i>	Long object name	ObjectString
<i>SelectionString</i>	Selection string	SelectionString
<i>ResObjectString</i>	Resolved long object name	ResObjectString
<i>ResolvedType</i>	Resolved object type	ResolvedType

Overview for MQOD

Availability: All WebSphere MQ systems, plus WebSphere MQ clients connected to those systems.

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

- Queue or distribution list
- Namelist
- Process definition
- Queue manager
- Topic

The structure is an input/output parameter on the MQOPEN and MQPUT1 calls.

Version: The current version of MQOD is MQOD_VERSION_4. Applications that you want to port between several environments must ensure that the required version of MQOD is supported in all the environments concerned. Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions that follow.

The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQOD that is supported by the environment, but with the initial value of the *Version* field set to MQOD_VERSION_1. To use fields that are not present in the version-1 structure, the application must set the *Version* field to the version number of the version required.

To open a distribution list, *Version* must be MQOD_VERSION_2 or greater.

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

Fields for MQOD

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

AlternateSecurityId (MQBYTE40)

This is a security identifier that is passed with the *AlternateUserId* to the authorization service to allow appropriate authorization checks to be performed. *AlternateSecurityId* is used only if:

- MQOO_ALTERNATE_USER_AUTHORITY is specified on the MQOPEN call, or
- MQPMO_ALTERNATE_USER_AUTHORITY is specified on the MQPUT1 call, *and* the *AlternateUserId* field is not entirely blank up to the first null character or the end of the field.

On Windows, *AlternateSecurityId* can be used to supply the Windows security identifier (SID) that uniquely identifies the *AlternateUserId*. The SID for a user can be obtained from the Windows system by use of the `LookupAccountName()` Windows API call.

On z/OS, this field is ignored.

The *AlternateSecurityId* field has the following structure:

- The first byte is a binary integer containing the length of the significant data that follows; the value excludes the length byte itself. If no security identifier is present, the length is zero.
- The second byte indicates the type of security identifier that is present; the following values are possible:
 - MQSIDT_NT_SECURITY_ID**
Windows security identifier.
 - MQSIDT_NONE**
No security identifier.
- The third and subsequent bytes up to the length defined by the first byte contain the security identifier itself.
- Remaining bytes in the field are set to binary zero.

You can use the following special value:

MQSID_NONE

No security identifier specified.

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

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

This is an input field. The length of this field is given by `MQ_SECURITY_ID_LENGTH`. The initial value of this field is `MQSID_NONE`. This field is ignored if *Version* is less than `MQOD_VERSION_3`.

AlternateUserId (MQCHAR12)

If you specify `MQOO_ALTERNATE_USER_AUTHORITY` for the MQOPEN call, or `MQPMO_ALTERNATE_USER_AUTHORITY` for the MQPUT1 call, this field

contains an alternate user identifier that is used to check the authorization for the open, in place of the user identifier that the application is currently running under. Some checks, however, are still carried out with the current user identifier (for example, context checks).

If MQOO_ALTERNATE_USER_AUTHORITY or MQPMO_ALTERNATE_USER_AUTHORITY is specified and this field is entirely blank up to the first null character or the end of the field, the open can succeed only if no user authorization is needed to open this object with the options specified.

If neither MQOO_ALTERNATE_USER_AUTHORITY nor MQPMO_ALTERNATE_USER_AUTHORITY is specified, this field is ignored.

The following differences exist in the environments indicated:

- On z/OS, only the first 8 characters of *AlternateUserId* are used to check the authorization for the open. However, the current user identifier must be authorized to specify this particular alternate user identifier; all 12 characters of the alternate user identifier are used for this check. The user identifier must contain only characters allowed by the external security manager.

If *AlternateUserId* is specified for a queue, the value can be used subsequently by the queue manager when messages are put. If the MQPMO_*_CONTEXT options specified on the MQPUT or MQPUT1 call cause the queue manager to generate the identity context information, the queue manager places the *AlternateUserId* into the *UserIdentifier* field in the MQMD of the message, in place of the current user identifier.

- In other environments, *AlternateUserId* is used only for access control checks on the object being opened. If the object is a queue, *AlternateUserId* does not affect the content of the *UserIdentifier* field in the MQMD of messages sent using that queue handle.

This is an input field. The length of this field is given by MQ_USER_ID_LENGTH. The initial value of this field is the null string in C, and 12 blank characters in other programming languages.

DynamicQName (MQCHAR48)

This is the name of a dynamic queue that is to be created by the MQOPEN call. This is of relevance only when *ObjectName* specifies the name of a model queue; in all other cases *DynamicQName* is ignored.

The characters that are valid in the name are the same as those for *ObjectName* (see above), except that an asterisk is also valid (see below). A name that is completely blank (or one in which only blanks appear before the first null character) is not valid if *ObjectName* is the name of a model queue.

If the last nonblank character in the name is an asterisk (*), the queue manager replaces the asterisk with a string of characters that guarantees that the name generated for the queue is unique at the local queue manager. To allow a sufficient number of characters for this, the asterisk is valid only in positions 1 through 33. There must be no characters other than blanks or a null character following the asterisk.

It is valid for the asterisk to appear in the first character position, in which case the name consists solely of the characters generated by the queue manager.

On z/OS, do not use a name with the asterisk in the first character position, as there can be no security checks made on a queue whose full name is generated automatically.

This is an input field. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is determined by the environment:

- On z/OS, the value is 'CSQ.*'.
- On other platforms, the value is 'AMQ.*'.

The value is a null-terminated string in C, and a blank-padded string in other programming languages.

InvalidDestCount (MQLONG)

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 that is not in a distribution list.

Note: If present, this field is set *only* if the *CompCode* parameter on the MQOPEN or MQPUT1 call is MQCC_OK or MQCC_WARNING; it is *not* set if the *CompCode* parameter is MQCC_FAILED.

This is an output field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_2.

KnownDestCount (MQLONG)

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 that is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_2.

ObjectName (MQCHAR48)

This is the local name of the object as defined on the queue manager identified by *ObjectQMgrName*. The name can contain the following characters:

- Uppercase alphabetic characters (A through Z)
- Lowercase alphabetic characters (a through z)
- Numeric digits (0 through 9)
- Period (.), forward slash (/), underscore (_), percent (%)

The name must not contain leading or embedded blanks, but can contain trailing blanks. Use a null character to indicate the end of significant data in the name; the null and any characters following it are treated as blanks. The following restrictions apply in the environments indicated:

- On systems that use EBCDIC Katakana, lowercase characters cannot be used.
- On z/OS:
 - Avoid names that begin or end with an underscore; they cannot be processed by the operations and control panels.

- The percent character has a special meaning to RACF. If RACF is used as the external security manager, names must not contain the percent. If they do, those names are not included in any security checks when RACF generic profiles are used.
- On i5/OS, names containing lowercase characters, forward slash, or percent, must be enclosed in quotation marks when specified on commands. These quotation marks must not be specified for names that occur as fields in structures or as parameters on calls.

The full topic name can be built from two different fields: *ObjectName* and *ObjectString*. For details of how these two fields are used, see “Using topic strings” on page 351.

The following points apply to the types of object indicated:

- If *ObjectName* is the name of a model queue, the queue manager creates a dynamic queue with the attributes of the model queue, and returns in the *ObjectName* field the name of the queue created. A model queue can be specified only on the MQOPEN call; a model queue is not valid on the MQPUT1 call.
- If *ObjectName* is the name of an alias queue with TARGTYPE(TOPIC), a security check is first made on the named alias queue; this is normal when alias queues are used. When the security check completes successfully, the MQOPEN call will continue and will behave like an MQOPEN call on an MQOT_TOPIC; this includes making a security check against the administrative topic object.
- If *ObjectName* and *ObjectQMgrName* identify a shared queue owned by the queue-sharing group to which the local queue manager belongs, there must not also be a queue definition of the same name on the local queue manager. If there is such a definition (a local queue, alias queue, remote queue, or model queue), the call fails with reason code MQRC_OBJECT_NOT_UNIQUE.
- If the object being opened is a distribution list (that is, *RecsPresent* is present and greater than zero), *ObjectName* must be blank or the null string. If this condition is not satisfied, the call fails with reason code MQRC_OBJECT_NAME_ERROR.
- If *ObjectType* is MQOT_Q_MGR, special rules apply; in this case the name must be entirely blank up to the first null character or the end of the field.

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

ObjectQMgrName (MQCHAR48)

This is the name of the queue manager on which the *ObjectName* object is defined. The characters that are valid in the name are the same as those for *ObjectName* (see above). A name that is entirely blank up to the first null character or the end of the field denotes the queue manager to which the application is connected (the local queue manager).

The following points apply to the types of object indicated:

- If *ObjectType* is MQOT_TOPIC, MQOT_NAMELIST, MQOT_PROCESS, or MQOT_Q_MGR, *ObjectQMgrName* must be blank or the name of the local queue manager.
- If *ObjectName* is the name of a model queue, the queue manager creates a dynamic queue with the attributes of the model queue, and returns in the

| *ObjectQMgrName* field the name of the queue manager on which the queue is
| created; this is the name of the local queue manager. A model queue can be
| specified only on the MQOPEN call; a model queue is not valid on the MQPUT1
| call.

- | • If *ObjectName* is the name of a cluster queue, and *ObjectQMgrName* is blank, the
| destination of messages sent using the queue handle returned by the MQOPEN
| call is chosen by the queue manager (or cluster workload exit, if one is installed)
| as follows:
 - | – If MQOO_BIND_ON_OPEN is specified, the queue manager selects a
| particular instance of the cluster queue while processing the MQOPEN call,
| and all messages put using this queue handle are sent to that instance.
 - | – If MQOO_BIND_NOT_FIXED is specified, the queue manager can choose a
| different instance of the destination queue (residing on a different queue
| manager in the cluster) for each successive MQPUT call that uses this queue
| handle.

| If the application needs to send a message to a *specific* instance of a cluster
| queue (that is, a queue instance that resides on a particular queue manager in
| the cluster), the application must specify the name of that queue manager in the
| *ObjectQMgrName* field. This forces the local queue manager to send the message
| to the specified destination queue manager.

- | • If *ObjectName* is the name of a shared queue that is owned by the queue-sharing
| group to which the local queue manager belongs, *ObjectQMgrName* can be the
| name of the queue-sharing group, the name of the local queue manager, or
| blank; the message is placed on the same queue whichever of these values is
| specified.

| Queue-sharing groups are supported only on z/OS.

- | • If *ObjectName* is the name of a shared queue that is owned by a remote
| queue-sharing group (that is, a queue-sharing group to which the local queue
| manager does *not* belong), *ObjectQMgrName* must be the name of the
| queue-sharing group. You can use the name of a queue manager that belongs to
| that group, but this can delay the message if that particular queue manager is
| not available when the message arrives at the queue-sharing group.
- | • If the object being opened is a distribution list (that is, *RecsPresent* is greater
| than zero), *ObjectQMgrName* must be blank or the null string. If this condition is
| not satisfied, the call fails with reason code
| MQRC_OBJECT_Q_MGR_NAME_ERROR.

| This is an input/output field for the MQOPEN call when *ObjectName* is the name
| of a model queue, and an input-only field in all other cases. The length of this field
| is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the
| null string in C, and 48 blank characters in other programming languages.

| **ObjectRecOffset (MQLONG)**

| This is the offset in bytes of the first MQOR object record from the start of the
| MQOD structure. The offset can be positive or negative. *ObjectRecOffset* is used
| only when a distribution list is being opened. The field is ignored if *RecsPresent* is
| zero.

| When a distribution list is being opened, an array of one or more MQOR object
| records must be provided in order to specify the names of the destination queues
| in the distribution list. This can be done in one of two ways:

- | • By using the offset field *ObjectRecOffset*.

In this case, the application must declare its own structure containing an MQOD followed by the array of MQOR records (with as many array elements as are needed), and set *ObjectRecOffset* to the offset of the first element in the array from the start of the MQOD. Ensure that this offset is correct and has a value that can be accommodated within an MQLONG (the most restrictive programming language is COBOL, for which the valid range is -999 999 999 through +999 999 999).

Use *ObjectRecOffset* for programming languages that do not support the pointer data type, or that implement the pointer data type in a way that is not portable to different environments (for example, the COBOL programming language).

- By using the pointer field *ObjectRecPtr*.

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

Use *ObjectRecPtr* for programming languages that support the pointer data type in a way that is portable to different environments (for example, the C programming language).

Whatever technique you choose, use one of *ObjectRecOffset* and *ObjectRecPtr*; the call fails with reason code MQRC_OBJECT_RECORDS_ERROR if both are zero, or both are nonzero.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_2.

ObjectRecPtr (MQPTR)

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

You can use either *ObjectRecPtr* or *ObjectRecOffset* to specify the object records, but not both; see the description of the *ObjectRecOffset* field above for details. If you do not use *ObjectRecPtr*, set it to the null pointer or null bytes.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQOD_VERSION_2.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

ObjectString (MQCHARV)

The ObjectString field specifies the long object name.

This specifies the long object name to be used. This field is only referenced for certain values of *ObjectType*, and is ignored for all other values. See the description of *ObjectType* for details of which values indicate that this field is used.

If *ObjectString* is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code MQRC_OBJECT_STRING_ERROR.

This is an input field. The initial values of the fields in this structure are the same as those in the MQCHARV structure.

The full topic name can be built from two different fields: *ObjectName* and *ObjectString*. For details of how these two fields are used, see “Using topic strings” on page 351.

ObjectType (MQLONG)

The type of object being named in the object descriptor. Possible values are:

MQOT_Q

Queue. The name of the object is found in the *ObjectName* field.

MQOT_NAMELIST

Namelist. The name of the object is found in the *ObjectName* field

MQOT_PROCESS

Process definition. The name of the object is found in the *ObjectName* field

MQOT_Q_MGR

Queue manager. The name of the object is found in the *ObjectName* field

MQOT_TOPIC

Topic. The full topic name can be built from two different fields: *ObjectName* and *ObjectString*.

For details of how those two fields are used, see “Using topic strings” on page 351.

If the object identified by the *ObjectName* field cannot be found, the call will fail with reason code MQRC_TOPIC_STRING_ERROR even if there is a string specified in *ObjectString*.

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

RecsPresent (MQLONG)

This is the number of MQOR object records that have been provided by the application. If this number is greater than zero, it indicates that a distribution list is being opened, with *RecsPresent* being the number of destination queues in the list. A distribution list can contain only one destination.

The value of *RecsPresent* must not be less than zero, and if it is greater than zero *ObjectType* must be MQOT_Q; the call fails with reason code MQRC_RECS_PRESENT_ERROR if these conditions are not satisfied.

On z/OS, this field must be zero.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_2.

ResObjectString (MQCHARV)

The ResObjectString field is the long object name after the queue manager resolves the name provided in the *ObjectName* field.

This field is returned only for topics and queue aliases that reference a topic object.

If the long object name is provided in *ObjectString* and nothing is provided in *ObjectName*, then the value returned in this field is the same as provided in *ObjectString*.

If this field is omitted (that is *ResObjectString.VSBufSize* is zero) then the *ResObjectString* will not be returned, but the length will be returned in *ResObjectString.VSLength*.

If the buffer length (provided in *ResObjectString.VSBufSize*) is shorter than the full *ResObjectString*, the string will be truncated and will return as many of the rightmost characters as can fit in the provided buffer.

If *ResObjectString* is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code MQRC_RES_OBJECT_STRING_ERROR.

ResolvedQMgrName (MQCHAR48)

This is the name of the destination queue manager after the local queue manager resolves the name. The name returned is the name of the queue manager that owns the queue identified by *ResolvedQName*. *ResolvedQMgrName* can be the name of the local queue manager.

If *ResolvedQName* is a shared queue that is owned by the queue-sharing group to which the local queue manager belongs, *ResolvedQMgrName* is the name of the queue-sharing group. If the queue is owned by some other queue-sharing group, *ResolvedQName* can be the name of the queue-sharing group or the name of a queue manager that is a member of the queue-sharing group (the nature of the value returned is determined by the queue definitions that exist at the local queue manager).

A nonblank value is returned only if the object is a single queue opened for browse, input, or output (or any combination). If the object opened is any of the following, *ResolvedQMgrName* is set to blanks:

- Not a queue
- A queue, but not opened for browse, input, or output
- A cluster queue with MQOO_BIND_NOT_FIXED specified (or with MQOO_BIND_AS_Q_DEF in effect when the *DefBind* queue attribute has the value MQBND_BIND_NOT_FIXED)
- A distribution list

This is an output field. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages. This field is ignored if *Version* is less than MQOD_VERSION_3.

ResolvedQName (MQCHAR48)

This is the name of the destination queue after the local queue manager resolves the name. The name returned is the name of a queue that exists on the queue manager identified by *ResolvedQMgrName*.

A nonblank value is returned only if the object is a single queue opened for browse, input, or output (or any combination). If the object opened is any of the following, *ResolvedQName* is set to blanks:

- Not a queue
- A queue, but not opened for browse, input, or output
- A distribution list
- An alias queue that references a topic object (refer to `ResObjectString` instead).
- An alias queue resolves to a topic object.

This is an output field. The length of this field is given by `MQ_Q_NAME_LENGTH`. The initial value of this field is the null string in C, and 48 blank characters in other programming languages. This field is ignored if *Version* is less than `MQOD_VERSION_3`.

ResolvedType (MQLONG)

The type of the resolved (base) object being opened.

The possible values are:

MQOT_Q

The resolved object is a queue. This value applies when a queue is opened directly or when an alias queue pointing to a queue is opened.

MQOT_TOPIC

The resolved object is a topic. This value applies when a topic is opened directly or when an alias queue pointing to a topic object is opened.

MQOT_NONE

The resolved type is neither a queue nor a topic.

ResponseRecOffset (MQLONG)

This is the offset in bytes of the first MQRR response record from the start of the MQOD structure. The offset can be positive or negative. *ResponseRecOffset* is used only when a distribution list is being opened. The field is ignored if *RecsPresent* is zero.

When a distribution list is being opened, you can provide an array of one or more MQRR response records in order to identify the queues that failed to open (*CompCode* field in MQRR), and the reason for each failure (*Reason* field in MQRR). The data is returned in the array of response records in the same order as the queue names occur in the array of object records. The queue manager sets the response records only when the outcome of the call is mixed (that is, some queues were opened successfully while others failed, or all failed but for different reasons); reason code `MQRC_MULTIPLE_REASONS` from the call indicates this case. If the same reason code applies to all queues, that reason is returned in the *Reason* parameter of the MQOPEN or MQPUT1 call, and the response records are not set. Response records are optional, but if they are supplied there must be *RecsPresent* of them.

The response records can be provided in the same way as the object records, either by specifying an offset in *ResponseRecOffset*, or by specifying an address in *ResponseRecPtr*; see the description of *ObjectRecOffset* above for details of how to do this. However, no more than one of *ResponseRecOffset* and *ResponseRecPtr* can be used; the call fails with reason code `MQRC_RESPONSE_RECORDS_ERROR` if both are nonzero.

For the MQPUT1 call, these response records are used to return information about errors that occur when the message is sent to the queues in the distribution list, as well as errors that occur when the queues are opened. The completion code and

reason code from the put operation for a queue replace those from the open operation for that queue only if the completion code from the latter was MQCC_OK or MQCC_WARNING.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_2.

ResponseRecPtr (MQPTR)

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

Use either *ResponseRecPtr* or *ResponseRecOffset* to specify the response records, but not both; see the description of the *ResponseRecOffset* field above for details. If you do not use *ResponseRecPtr*, set it to the null pointer or null bytes.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQOD_VERSION_2.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

SelectionString (MQCHARV)

This is the string used to provide the selection criteria used when retrieving messages off a queue.

SelectionString must not be provided in the following cases:

- If *ObjectType* is not MQOT_Q
- If the queue being opened is not being opened using one of the MQOO_BROWSE, or MQOO_INPUT_* options

If *SelectionString* is provided in these cases, the call fails with reason code MQRC_SELECTOR_INVALID_FOR_TYPE.

If *SelectionString* is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code MQRC_SELECTION_STRING_ERROR.

StruId (MQCHAR4)

This is the structure identifier; the value must be:

MQOD_STRUC_ID

Identifier for object descriptor structure.

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

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

UnknownDestCount (MQLONG)

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 that is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_2.

Version (MQLONG)

This is the structure version number; the value must be one of the following:

MQOD_VERSION_1

Version-1 object descriptor structure.

MQOD_VERSION_2

Version-2 object descriptor structure.

MQOD_VERSION_3

Version-3 object descriptor structure.

MQOD_VERSION_4

Version-4 object descriptor structure.

All versions are supported in all WebSphere MQ V7.0 environments.

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

MQOD_CURRENT_VERSION

Current version of object descriptor structure.

This is always an input field. The initial value of this field is MQOD_VERSION_1.

Initial values and language declarations for MQOD

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQOD_STRUC_ID	'0dbb'
<i>Version</i>	MQOD_VERSION_1	1
<i>ObjectType</i>	MQOT_Q	1
<i>ObjectName</i>	None	Null string or blanks
<i>ObjectQMgrName</i>	None	Null string or blanks
<i>DynamicQName</i>	None	'CSQ.*' on z/OS; 'AMQ.*' otherwise
<i>AlternateUserId</i>	None	Null string or blanks
<i>RecsPresent</i>	None	0
<i>KnownDestCount</i>	None	0
<i>UnknownDestCount</i>	None	0
<i>InvalidDestCount</i>	None	0
<i>ObjectRecOffset</i>	None	0
<i>ResponseRecOffset</i>	None	0
<i>ObjectRecPtr</i>	None	Null pointer or null bytes
<i>ResponseRecPtr</i>	None	Null pointer or null bytes
<i>AlternateSecurityId</i>	MQSID_NONE	Nulls
<i>ResolvedQName</i>	None	Null string or blanks
<i>ResolvedQMgrName</i>	None	Null string or blanks

Field name	Name of constant	Value of constant
<i>ObjectString</i>	MQCHARV_DEFAULT	As defined for MQCHARV
<i>SelectionString</i>	MQCHARV_DEFAULT	As defined for MQCHARV
<i>ResObjectString</i>	MQCHARV_DEFAULT	As defined for MQCHARV
<i>ResolvedType</i>	MQOT_NONE	0
Notes:		
<ol style="list-style-type: none"> 1. The symbol <code>b</code> represents a single blank character. 2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages. 3. In the C programming language, the macro variable <code>MQOD_DEFAULT</code> contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure: <pre>MQOD MyOD = {MQOD_DEFAULT};</pre> 		

C declaration

```
typedef struct tagMQOD MQOD;
struct tagMQOD {
    MQCHAR4    StrucId;           /* Structure identifier */
    MQLONG     Version;          /* Structure version number */
    MQLONG     ObjectType;       /* Object type */
    MQCHAR48   ObjectName;       /* Object name */
    MQCHAR48   ObjectQMgrName;   /* Object queue manager name */
    MQCHAR48   DynamicQName;     /* Dynamic queue name */
    MQCHAR12   AlternateUserId;  /* Alternate user identifier */
    /* Ver:1 */
    MQLONG     RecsPresent;       /* Number of object records present */
    MQLONG     KnownDestCount;   /* Number of local queues opened
    successfully */
    MQLONG     UnknownDestCount; /* Number of remote queues opened
    successfully */
    MQLONG     InvalidDestCount; /* Number of queues that failed to
    open */
    MQLONG     ObjectRecOffset;  /* Offset of first object record from
    start of MQOD */
    MQLONG     ResponseRecOffset; /* Offset of first response record
    from start of MQOD */
    MQPTR      ObjectRecPtr;     /* Address of first object record */
    MQPTR      ResponseRecPtr;   /* Address of first response record */
    /* Ver:2 */
    MQBYTE40   AlternateSecurityId; /* Alternate security identifier */
    MQCHAR48   ResolvedQName;     /* Resolved queue name */
    MQCHAR48   ResolvedQMGrName; /* Resolved queue manager name */
    /* Ver:3 */
    MQCHARV    ObjectString;      /* Object Long name */
    MQCHARV    SelectionString;   /* Message Selector */
    MQCHARV    ResObjectString;   /* Resolved Long object name*/
    MQLONG     ResolvedType       /* Alias queue resolved
    object type */
    /* Ver:4 */
};
```

COBOL declaration

```
** MQOD structure
   10 MQOD.
** Structure identifier
   15 MQOD-STRUCID                PIC X(4).
** Structure version number
```

15	MQOD-VERSION	PIC S9(9) BINARY.
**	Object type	
15	MQOD-OBJECTTYPE	PIC S9(9) BINARY.
**	Object name	
15	MQOD-OBJECTNAME	PIC X(48).
**	Object queue manager name	
15	MQOD-OBJECTQMGRNAME	PIC X(48).
**	Dynamic queue name	
15	MQOD-DYNAMICQNAME	PIC X(48).
**	Alternate user identifier	
15	MQOD-ALTERNATEUSERID	PIC X(12).
**	Number of object records present	
15	MQOD-RECSPRESENT	PIC S9(9) BINARY.
**	Number of local queues opened successfully	
15	MQOD-KNOWNDDESTCOUNT	PIC S9(9) BINARY.
**	Number of remote queues opened successfully	
15	MQOD-UNKNOWNDDESTCOUNT	PIC S9(9) BINARY.
**	Number of queues that failed to open	
15	MQOD-INVALIDDESTCOUNT	PIC S9(9) BINARY.
**	Offset of first object record from start of MQOD	
15	MQOD-OBJECTRECOFFSET	PIC S9(9) BINARY.
**	Offset of first response record from start of MQOD	
15	MQOD-RESPONSERECOFFSET	PIC S9(9) BINARY.
**	Address of first object record	
15	MQOD-OBJECTRECPTTR	POINTER.
**	Address of first response record	
15	MQOD-RESPONSERECPTTR	POINTER.
**	Alternate security identifier	
15	MQOD-ALTERNATESECURITYID	PIC X(40).
**	Resolved queue name	
15	MQOD-RESOLVEDQNAME	PIC X(48).
**	Resolved queue manager name	
15	MQOD-RESOLVEDQMGRNAME	PIC X(48).
**	Object Long name	
15	MQOD-OBJECTSTRING.	
**	Address of variable length string	
20	MQOD-OBJECTSTRING-VSPTR	POINTER.
**	Offset of variable length string	
20	MQOD-OBJECTSTRING-VSOFFSET	PIC S9(9) BINARY.
**	size of buffer	
20	MQOD-OBJECTSTRING-VSBUFSIZE	PIC S9(9) BINARY.
**	Length of variable length string	
20	MQOD-OBJECTSTRING-VSLENGTH	PIC S9(9) BINARY.
**	CCSID of variable length string	
20	MQOD-OBJECTSTRING-VSCCSID	PIC S9(9) BINARY.
**	Message Selector	
15	MQOD-SELECTIONSTRING.	
**	Address of variable length string	
20	MQOD-SELECTIONSTRING-VSPTR	POINTER.
**	Offset of variable length string	
20	MQOD-SELECTIONSTRING-VSOFFSET	PIC S9(9) BINARY.
**	size of buffer	
20	MQOD-SELECTIONSTRING-VSBUFSIZE	PIC S9(9) BINARY.
**	Length of variable length string	
20	MQOD-SELECTIONSTRING-VSLENGTH	PIC S9(9) BINARY.
**	CCSID of variable length string	
20	MQOD-SELECTIONSTRING-VSCCSID	PIC S9(9) BINARY.
**	Resolved Long object name	
15	MQOD-RESOBJECTSTRING.	
**	Address of variable length string	
20	MQOD-RESOBJECTSTRING-VSPTR	POINTER.
**	Offset of variable length string	
20	MQOD-RESOBJECTSTRING-VSOFFSET	PIC S9(9) BINARY.
**	size of buffer	
20	MQOD-RESOBJECTSTRING-VSBUFSIZE	PIC S9(9) BINARY.
**	Length of variable length string	
20	MQOD-RESOBJECTSTRING-VSLENGTH	PIC S9(9) BINARY.

```

** CCSID of variable length string
   20 MQOD-RESOBJECTSTRING-VSCCSID PIC S9(9) BINARY.
** Alias queue resolved object type
   15 MQOD-RESOLVEDTYPE PIC S9(9) BINARY.

```

PL/I declaration

```

dcl
  1 MQOD based,
    3 StrucId          char(4),          /* Structure identifier */
    3 Version         fixed bin(31),    /* Structure version number */
    3 ObjectType      fixed bin(31),    /* Object type */
    3 ObjectName     char(48),          /* Object name */
    3 ObjectQMgrName char(48),          /* Object queue manager name */
    3 DynamicQName   char(48),          /* Dynamic queue name */
    3 AlternateUserId char(12),         /* Alternate user identifier */
    3 RecsPresent    fixed bin(31),    /* Number of object records
                                           present */
    3 KnownDestCount fixed bin(31),    /* Number of local queues opened
                                           successfully */
    3 UnknownDestCount fixed bin(31), /* Number of remote queues opened
                                           successfully */
    3 InvalidDestCount fixed bin(31), /* Number of queues that failed to
                                           open */
    3 ObjectRecOffset fixed bin(31), /* Offset of first object record
                                           from start of MQOD */
    3 ResponseRecOffset fixed bin(31), /* Offset of first response record
                                           from start of MQOD */
    3 ObjectRecPtr    pointer,          /* Address of first object record */
    3 ResponseRecPtr pointer,          /* Address of first response
                                           record */
    3 AlternateSecurityId char(40),    /* Alternate security identifier */
    3 ResolvedQName   char(48),          /* Resolved queue name */
    3 ResolvedQMgrName char(48),          /* Resolved queue manager name */
    3 ObjectString,    /* Object Long name */
      5 VSPtr          pointer,          /* Address of variable length string */
      5 VSOffset      fixed bin(31),    /* Offset of variable length string */
      5 VSBufSize     fixed bin(31),    /* size of buffer */
      5 VSLength      fixed bin(31),    /* Length of variable length string */
      5 VSCCSID       fixed bin(31),    /* CCSID of variable length string */
    3 SelectionString, /* Message Selection */
      5 VSPtr          pointer,          /* Address of variable length string */
      5 VSOffset      fixed bin(31),    /* Offset of variable length string */
      5 VSBufSize     fixed bin(31),    /* size of buffer */
      5 VSLength      fixed bin(31),    /* Length of variable length string */
      5 VSCCSID       fixed bin(31),    /* CCSID of variable length string */
    3 ResObjectString, /* Resolved Long object name */
      5 VSPtr          pointer,          /* Address of variable length string */
      5 VSOffset      fixed bin(31),    /* Offset of variable length string */
      5 VSBufSize     fixed bin(31),    /* size of buffer */
      5 VSLength      fixed bin(31),    /* Length of variable length string */
      5 VSCCSID       fixed bin(31),    /* CCSID of variable length string */
    3 ResolvedType    fixed bin(31); /* Alias queue resolved object type */

```

System/390 assembler declaration

```

MQOD          DSECT
MQOD_STRUCID  DS   CL4  Structure identifier
MQOD_VERSION  DS   F    Structure version number
MQOD_OBJECTTYPE DS   F    Object type
MQOD_OBJECTNAME DS   CL48 Object name
MQOD_OBJECTQMGRNAME DS   CL48 Object queue manager name
MQOD_DYNAMICQNAME DS   CL48 Dynamic queue name
MQOD_ALTERNATEUSERID DS   CL12 Alternate user identifier
MQOD_RECSPRESENT DS   F    Number of object records present
MQOD_KNOWNDSTCOUNT DS   F    Number of local queues opened
*              successfully
MQOD_UNKNOWNDESTCOUNT DS   F    Number of remote queues opened

```

	*			successfully
MQOD_INVALIDDESTCOUNT	DS	F		Number of queues that failed to open
	*			
MQOD_OBJECTRECOFFSET	DS	F		Offset of first object record from start of MQOD
	*			
MQOD_RESPONSERECOFFSET	DS	F		Offset of first response record from start of MQOD
	*			
MQOD_OBJECTRECPtr	DS	F		Address of first object record
MQOD_RESPONSERECPtr	DS	F		Address of first response record
MQOD_ALTERNATESECURITYID	DS	XL40		Alternate security identifier
MQOD_RESOLVEDQNAME	DS	CL48		Resolved queue name
MQOD_RESOLVEDQMGRNAME	DS	CL48		Resolved queue manager name
MQOD_OBJECTSTRING	DS	F		Object Long name
MQOD_OBJECTSTRING_VSPTR	DS	F		Address of variable length string
MQOD_OBJECTSTRING_VSOFFSET	DS	F		Offset of variable length string
MQOD_OBJECTSTRING_VSBUFSIZE	DS	F		size of buffer
MQOD_OBJECTSTRING_VSLENGTH	DS	F		Length of variable length string
MQOD_OBJECTSTRING_VSCCSID	DS	F		CCSID of variable length string
MQOD_OBJECTSTRING_LENGTH	EQU		*-	MQOD_OBJECTSTRING
	ORG			MQOD_OBJECTSTRING
MQOD_OBJECTSTRING_AREA	DS			CL(MQOD_OBJECTSTRING_LENGTH)
	*			
MQOD_SELECTIONSTRING	DS	F		Message Selector
MQOD_SELECTIONSTRING_VSPTR	DS	F		Address of variable length string
MQOD_SELECTIONSTRING_VSOFFSET	DS	F		Offset of variable length string
MQOD_SELECTIONSTRING_VSBUFSIZE	DS	F		size of buffer
MQOD_SELECTIONSTRING_VSLENGTH	DS	F		Length of variable length string
MQOD_SELECTIONSTRING_VSCCSID	DS	F		CCSID of variable length string
MQOD_SELECTIONSTRING_LENGTH	EQU		*-	MQOD_SELECTIONSTRING
	ORG			MQOD_SELECTIONSTRING
MQOD_SELECTIONSTRING_AREA	DS			CL(MQOD_SELECTIONSTRING_LENGTH)
	*			
MQOD_RESOBJECTSTRING	DS	F		Resolved Long object name
MQOD_RESOBJECTSTRING_VSPTR	DS	F		Address of variable length string
MQOD_RESOBJECTSTRING_VSOFFSET	DS	F		Offset of variable length string
MQOD_RESOBJECTSTRING_VSBUFSIZE	DS	F		size of buffer
MQOD_RESOBJECTSTRING_VSLENGTH	DS	F		Length of variable length string
MQOD_RESOBJECTSTRING_VSCCSID	DS	F		CCSID of variable length string
MQOD_RESOBJECTSTRING_LENGTH	EQU		*-	MQOD_RESOBJECTSTRING
	ORG			MQOD_RESOBJECTSTRING
MQOD_RESOBJECTSTRING_AREA	DS			CL(MQOD_RESOBJECTSTRING_LENGTH)
MQOD_RESOLVEDTYPE	DS	F		Alias queue object resolved type
	*			
MQOD_LENGTH	EQU		*-	MQOD
	ORG			MQOD
MQOD_AREA	DS			CL(MQOD_LENGTH)

Visual Basic declaration

Type MQOD

StrucId	As String*4	'Structure identifier'
Version	As Long	'Structure version number'
ObjectType	As Long	'Object type'
ObjectName	As String*48	'Object name'
ObjectQMgrName	As String*48	'Object queue manager name'
DynamicQName	As String*48	'Dynamic queue name'
AlternateUserId	As String*12	'Alternate user identifier'
RecsPresent	As Long	'Number of object records present'
KnownDestCount	As Long	'Number of local queues opened' 'successfully'
UnknownDestCount	As Long	'Number of remote queues opened' 'successfully'
InvalidDestCount	As Long	'Number of queues that failed to' 'open'
ObjectRecOffset	As Long	'Offset of first object record from' 'start of MQOD'
ResponseRecOffset	As Long	'Offset of first response record' 'from start of MQOD'

```

ObjectRecPtr      As MQPTR      'Address of first object record'
ResponseRecPtr    As MQPTR      'Address of first response record'
AlternateSecurityId As MQBYTE40  'Alternate security identifier'
ResolvedQName     As String*48  'Resolved queue name'
ResolvedQMgrName  As String*48  'Resolved queue manager name'
End Type

```

MQOR – Object record

The following table summarizes the fields in the structure.

Table 53. Fields in MQOR

Field	Description	Topic
<i>ObjectName</i>	Object name	ObjectName
<i>ObjectQMgrName</i>	Object queue manager name	ObjectQMgrName

Overview for MQOR

Availability: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Purpose: Use the MQOR structure to specify the queue name and queue-manager name of a single destination queue. MQOR is an input structure for the MQOPEN and MQPUT1 calls.

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

Usage: By providing an array of these structures on the MQOPEN call, you can open a list of queues; this list is called a *distribution list*. Each message put using the queue handle returned by that MQOPEN call is placed on each of the queues in the list, provided that the queue was opened successfully.

Fields for MQOR

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

ObjectName (MQCHAR48)

This is the same as the *ObjectName* field in the MQOD structure (see MQOD for details), except that:

- It must be the name of a queue.
- It must not be the name of a model queue.

This is always an input field. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ObjectQMgrName (MQCHAR48)

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

This is always an input field. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

Initial values and language declarations for MQOR

Table 54. Initial values of fields in MQOR for MQOR

Field name	Name of constant	Value of constant
<i>ObjectName</i>	None	Null string or blanks
<i>ObjectQMgrName</i>	None	Null string or blanks
<p>Notes:</p> <ol style="list-style-type: none"> 1. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages. 2. In the C programming language, the macro variable MQOR_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure: MQOR MyOR = {MQOR_DEFAULT}; 		

C declaration

```
typedef struct tagMQOR MQOR;
struct tagMQOR {
    MQCHAR48 ObjectName;      /* Object name */
    MQCHAR48 ObjectQMgrName; /* Object queue manager name */
};
```

COBOL declaration

```
** MQOR structure
10 MQOR.
** Object name
15 MQOR-OBJECTNAME PIC X(48).
** Object queue manager name
15 MQOR-OBJECTQMGRNAME PIC X(48).
```

PL/I declaration

```
dcl
1 MQOR based,
3 ObjectName char(48), /* Object name */
3 ObjectQMgrName char(48); /* Object queue manager name */
```

Visual Basic declaration

```
Type MQOR
    ObjectName As String*48 'Object name'
    ObjectQMgrName As String*48 'Object queue manager name'
End Type
```

MQPD – Property descriptor

The following table summarizes the fields in the structure.

Table 55. Fields in MQPD

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>Options</i>	Options	Options

Table 55. Fields in MQPD (continued)

Field	Description	Topic
<i>Support</i>	Required support for message property	Support
<i>Context</i>	Message context to which property belongs	Context
<i>CopyOptions</i>	Copy options to which property belongs	CopyOptions

Overview for MQPD

Availability: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, z/OS and WebSphere MQ clients.

Purpose: The MQPD is used to define the attributes of a property. The structure is an input/output parameter on the MQSETMP call and an output parameter on the MQINQMP call.

Character set and encoding: Data in MQPD must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields for MQPD

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

Context (MQLONG)

This describes what message context the property belongs to.

When a queue manager receives a message containing a WebSphere MQ-defined property that the queue manager recognizes as being incorrect. the queue manager corrects the value of the *Context* field.

The following option can be specified:

MQPD_USER_CONTEXT

The property is associated with the user context.

No special authorization is required to be able to set a property associated with the user context using the MQSETMP call.

On a WebSphere MQ Version 7.0 queue manager, a property associated with the user context is saved as described for MQOO_SAVE_ALL_CONTEXT. An MQPUT call with MQPMO_PASS_ALL_CONTEXT specified, causes the property to be copied from the saved context into the new message.

If the option previously described is not required, the following option can be used:

MQPD_NO_CONTEXT

The property is not associated with a message context.

An unrecognized value is rejected with a *Reasoncode* of MQRC_PD_ERROR

This is an input/output field to the MQSETMP call and an output field from the MQINQMP call. The initial value of this field is MQPD_NO_CONTEXT.

CopyOptions (MQLONG)

This describes which type of messages the property should be copied into. This is an output only field for recognized WebSphere MQ-defined properties; WebSphere MQ sets the appropriate value.

When a queue manager receives a message containing a WebSphere MQ-defined property that the queue manager recognizes as being incorrect, the queue manager corrects the value of the *CopyOptions* field.

You can specify one or more of these options, and if you need more than one, the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

MQCOPY_FORWARD

This property is copied into a message being forwarded.

MQCOPY_PUBLISH

This property is copied into the message received by a subscriber when a message is being published.

MQCOPY_REPLY

This property is copied into a reply message.

MQCOPY_REPORT

This property is copied into a report message.

MQCOPY_ALL

This property is copied into all types of subsequent messages.

Default option: The following option can be specified to supply the default set of copy options:

MQCOPY_DEFAULT

This property is copied into a message being forwarded, into a report message, or into a message received by a subscriber when a message is being published.

This is equivalent to specifying the combination of options MQCOPY_FORWARD, plus MQCOPY_REPORT, plus MQCOPY_PUBLISH.

If none of the options described above is required, use the following option:

MQCOPY_NONE

Use this value to indicate that no other copy options have been specified; programmatically no relationship exists between this property and subsequent messages. This is always returned for message descriptor properties.

This is an input/output field to the MQSETMP call and an output field from the MQINQMP call. The initial value of this field is MQCOPY_DEFAULT.

Options (MQLONG)

The value must be:

MQPD_NONE

No options specified

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

StrucId (MQCHAR4)

This is the structure identifier; the value must be:

MQPD_STRUC_ID

Identifier for property descriptor structure.

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

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

Support (MQLONG)

This field describes what level of support for the message property is required of the queue manager, in order for the message containing this property to be put to a queue. This applies only to WebSphere MQ-defined properties; support for all other properties is optional.

The field is automatically set to the correct value when the WebSphere MQ-defined property is known by the queue manager. If the property is not recognized, MQPD_SUPPORT_OPTIONAL is assigned. When a queue manager receives a message containing a WebSphere MQ-defined property that the queue manager recognizes as being incorrect, the queue manager corrects the value of the *Support* field.

When setting a WebSphere MQ-defined property using the MQSETMP call on a message handle where the MQCMHO_NO_VALIDATION option was set, *Support* becomes an input field. This allows an application to put a WebSphere MQ-defined property, with the correct value, where the property is unsupported by the connected queue manager, but where the message is intended to be processed on another queue manager.

The value MQPD_SUPPORT_OPTIONAL is always assigned to properties that are not WebSphere MQ-defined properties.

If a WebSphere MQ Version 7.0 queue manager, that supports message properties, receives a property that contains an unrecognized *Support* value, the property is treated as if:

- MQPD_SUPPORT_REQUIRED was specified if any of the unrecognized values are contained in the MQPD_REJECT_UNSUP_MASK.
- MQPD_SUPPORT_REQUIRED_IF_LOCAL was specified if any of the unrecognized values are contained in the MQPD_ACCEPT_UNSUP_IF_XMIT_MASK
- MQPD_SUPPORT_OPTIONAL was specified otherwise.

One of the following values is returned by the MQINQMP call, or one of the values can be specified, when using the MQSETMP call on a message handle where the MQCMHO_NO_VALIDATION option is set:

MQPD_SUPPORT_OPTIONAL

The property is accepted by a queue manager even if it is not supported.

The property can be discarded in order for the message to flow to a queue

manager that does not support message properties. This value is also assigned to properties that are not WebSphere MQ-defined.

MQPD_SUPPORT_REQUIRED

Support for the property is required. The message is rejected by a queue manager that does not support the WebSphere MQ-defined property. The MQPUT or MQPUT1 call fails with completion code MQCC_FAILED and reason code MQRC_UNSUPPORTED_PROPERTY.

MQPD_SUPPORT_REQUIRED_IF_LOCAL

The message is rejected by a queue manager that does not support the WebSphere MQ-defined property if the message is destined for a local queue. The MQPUT or MQPUT1 call fails with completion code MQCC_FAILED and reason code MQRC_UNSUPPORTED_PROPERTY.

The MQPUT or MQPUT1 call succeeds if the message is destined for a remote queue manager.

This is an output field on the MQINQMP call and an input field on the MQSETMP call if the message handle was created with the MQCMHO_NO_VALIDATION option set. The initial value of this field is MQPD_SUPPORT_OPTIONAL.

Version (MQLONG)

This is the structure version number; the value must be:

MQPD_VERSION_1

Version-1 property descriptor structure.

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

MQPD_CURRENT_VERSION

Current version of property descriptor structure.

This is always an input field. The initial value of this field is MQPD_VERSION_1.

Initial values and language declarations for MQPD

Table 56. Initial values of fields in MQPD

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQPD_STRUC_ID	'PD'
<i>Version</i>	MQPD_VERSION_1	1
<i>Options</i>	MQPD_NONE	0
<i>Support</i>	MQPD_SUPPORT_OPTIONAL	0
<i>Context</i>	MQPD_NO_CONTEXT	0
<i>CopyOptions</i>	MQCOPY_DEFAULT	0

Notes:

1. In the C programming language, the macro variable MQPD_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQPD MyPD = {MQPD_DEFAULT};
```

C declaration

```
typedef struct tagMQPD MQPD;
struct tagMQPD {
    MQCHAR4  StrucId;      /* Structure identifier */
    MQLONG   Version;     /* Structure version number */
    MQLONG   Options;     /* Options that control the action of
                          MQSETMP and MQINQMP */
    MQLONG   Support;     /* Property support option */
    MQLONG   Context;     /* Property context */
    MQLONG   CopyOptions; /* Property copy options */
};
```

COBOL declaration

```
** MQPD structure
10 MQPD.
** Structure identifier
15 MQPD-STRUCID PIC X(4).
** Structure version number
15 MQPD-VERSION PIC S9(9) BINARY.
** Options that control the action of MQSETMP and
** MQINQMP
15 MQPD-OPTIONS PIC S9(9) BINARY.
** Property support option
15 MQPD-SUPPORT PIC S9(9) BINARY.
** Property context
15 MQPD-CONTEXT PIC S9(9) BINARY.
** Property copy options
15 MQPD-COPYOPTIONS PIC S9(9) BINARY.
```

PL/I declaration

```
dc1
1 MQPD based,
3 StrucId char(4), /* Structure identifier */
3 Version fixed bin(31), /* Structure version number */
3 Options fixed bin(31), /* Options that control the action
                          of MQSETMP and MQINQMP */
3 Support fixed bin(31), /* Property support option */
3 Context fixed bin(31), /* Property context */
3 CopyOptions fixed bin(31); /* Property copy options */
```

System/390 assembler declaration

```
MQPD          DSECT
MQPD_STRUCID  DS CL4 Structure identifier
MQPD_VERSION  DS F   Structure version number
MQPD_OPTIONS  DS F   Options that control the
*              action of MQSETMP and MQINQMP
MQPD_SUPPORT  DS F   Property support option
MQPD_CONTEXT  DS F   Property context
MQPD_COPYOPTIONS DS F Property copy options
MQPD_LENGTH   EQU *-MQPD
MQPD_AREA     DS CL(MQPD_LENGTH)
```

MQPMO – Put-message options

The following table summarizes the fields in the structure.

Table 57. MQPMO structure

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version

Table 57. MQPMO structure (continued)

Field	Description	Topic
<i>Options</i>	Options that control the action of MQPUT and MQPUT1	Options
<i>Timeout</i>	Reserved	Timeout
<i>Context</i>	Object handle of input queue	Context
<i>KnownDestCount</i>	Number of messages sent successfully to local queues	KnownDestCount
<i>UnknownDestCount</i>	Number of messages sent successfully to remote queues	UnknownDestCount
<i>InvalidDestCount</i>	Number of messages that could not be sent	InvalidDestCount
<i>ResolvedQName</i>	Resolved name of destination queue	ResolvedQName
<i>ResolvedQMgrName</i>	Resolved name of destination queue manager	ResolvedQMgrName
Note: The remaining fields are ignored if <i>Version</i> is less than MQPMO_VERSION_2.		
<i>RecsPresent</i>	Number of put message records or response records present	RecsPresent
<i>PutMsgRecFields</i>	Flags indicating which MQPMR fields are present	PutMsgRecFields
<i>PutMsgRecOffset</i>	Offset of first put-message record from start of MQPMO	PutMsgRecOffset
<i>ResponseRecOffset</i>	Offset of first response record from start of MQPMO	ResponseRecOffset
<i>PutMsgRecPtr</i>	Address of first put message record	PutMsgRecPtr
<i>ResponseRecPtr</i>	Address of first response record	ResponseRecPtr
Note: The remaining fields are ignored if <i>Version</i> is less than MQPMO_VERSION_3.		
<i>OriginalMsgHandle</i>	Original message handle	OriginalMsgHandle
<i>NewMsgHandle</i>	New message handle	NewMsgHandle
<i>Action</i>	Type of put being performed and the relationship between the original message specified by the <i>OriginalMsgHandle</i> field and the new message specified by the <i>NewMsgHandle</i> field	Action
<i>PubLevel</i>	Level of subscription targeted by the publication	PubLevel

Overview for MQPMO

Availability: All WebSphere MQ systems, plus WebSphere MQ clients connected to these systems.

Purpose: The MQPMO structure allows the application to specify options that control how messages are placed on queues, or published to topics. The structure is an input/output parameter on the MQPUT and MQPUT1 calls.

Version: The current version of MQPMO is MQPMO_VERSION_3. Certain fields are available only in certain versions of MQGPO. If you need to port applications between several environments, you must ensure that the version of MQPMO is

consistent across all environments. Fields that exist only in particular versions of the structure are identified as such in MQPMO - Put-message options and in the field descriptions.

The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQPMO that is supported by the environment, but with the initial value of the *Version* field set to MQPMO_VERSION_1. To use fields that are not present in the version-1 structure, the application must set the *Version* field to the version number of the version required.

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

Fields for MQPMO

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

Action (MQLONG)

This specifies the type of put being performed and the relationship between the original message specified by the OriginalMsgHandle field and the new message specified by the NewMsgHandle field. The properties of the message are chosen by the queue manager according to the value of the Action specified.

You can choose to supply the contents of the message descriptor using the MsgDesc parameter on the MQPUT or MQPUT1 calls. Alternatively it is possible not to supply the MsgDesc parameter, or to specify that it is output-only by including MQPMO_MD_FOR_OUTPUT_ONLY in the Options field of the MQPMO structure.

If the MsgDesc parameter is not supplied, or if it is specified to be output-only, then the message descriptor for the new message is populated from the message handle fields of the MQPMO, according to the rules described in this topic.

The context setting and passing activities described in *WebSphere MQ Application Programming Guide* take effect after the message descriptor has been composed.

If an incorrect action value is specified, the call fails with the reason code MQRC_ACTION_ERROR.

Any one of the following actions can be specified:

MQACTP_NEW

A new message is being put, and no relationship to a previous message is being specified by the program. The message descriptor is composed as follows:

- If a MsgDesc is supplied on the MQPUT or MQPUT1 call, and MQPMO_MD_FOR_OUTPUT_ONLY is not in the MQPMO.Options, this is used as the message descriptor unmodified.
- If a MsgDesc is not supplied, or MQPMO_MD_FOR_OUTPUT_ONLY is in the MQPMO.Options then the queue manager generates the message descriptor using a combination of properties from OriginalMsgHandle and

NewMsgHandle. Any message descriptor fields explicitly set on the new message handle take precedence over those in the original message handle.

Message data is taken from the MQPUT or MQPUT1 Buffer parameter.

MQACTP_FORWARD

A previously retrieved message is being forwarded. The original message handle specifies the message that was previously retrieved.

The new message handle specifies any modifications to the properties (including any in the message descriptor) in the original message handle.

The message descriptor is composed as follows:

- If a MsgDesc is supplied on the MQPUT or MQPUT1 call, and MQPMO_MD_FOR_OUTPUT_ONLY is not in the MQPMO.Options, this is used as the message descriptor unmodified.
- If a MsgDesc is not supplied, or MQPMO_MD_FOR_OUTPUT_ONLY is in the MQPMO.Options then the queue manager generates the message descriptor using a combination of properties from OriginalMsgHandle and NewMsgHandle. Any message descriptor fields explicitly set on the new message handle take precedence over those in the original message handle.
- If MQPMO_NEW_MSG_ID or MQPMO_NEW_CORREL_ID are specified in the MQPMO.Options, then these are honoured.

The message properties are composed as follows:

- All properties from the original message handle which have MQCOPY_FORWARD in the MQPD.CopyOptions
- All properties from the new message handle. For each property in the new message handle that has the same name as a property in the original message handle, the value is taken from the new message handle. The only exception to this rule is the special case when the property in the new message handle has the same name as a property in the original message handle, but the value of the property is null. In this case the property is removed from the message.

The message data to be forwarded is taken from the MQPUT or MQPUT1 Buffer parameter.

MQACTP_REPLY

A reply is being made to a previously retrieved message. The original message handle specifies the message that was previously retrieved.

The new message handle specifies any modifications to the properties (including any in the message descriptor) in the original message handle.

The message descriptor is composed as follows:

- If a MsgDesc is supplied on the MQPUT or MQPUT1 call, and MQPMO_MD_FOR_OUTPUT_ONLY is not in the MQPMO.Options, this is used as the message descriptor unmodified.
- If a MsgDesc is not supplied, or MQPMO_MD_FOR_OUTPUT_ONLY is in the MQPMO.Options then initial message descriptor fields are chosen as follows:

Table 58. Reply message handle transformation

Field in MQMD	Value used
Report	If MQRO_PASS_DISCARD_AND_EXPIRY and MQRO_DISCARD_MSG are set: MQRO_DISCARD_MSG otherwise MQRO_NONE
MsgType	MQMT_REPLY
Expiry	If MQRO_PASS_DISCARD_AND_EXPIRY is set: Copied from the input message otherwise MQEI_UNLIMITED
Feedback	MQFB_NONE
MsgId	If MQPMO_NEW_MSG_ID is set: A new message identifier is generated else if MQRO_PASS_MSG_ID is set: Copied from the input message otherwise MQMI_NONE
CorrelId	If MQPMO_NEW_CORREL_ID is set: A new correlation identifier is generated else if MQRO_COPY_MSG_ID_TO_CORREL_ID is set: Copied from the MsgId field of the input message else if MQRO_PASS_CORREL_ID is set: Copied from the CorrelId field of the input message otherwise MQCI_NONE
BackoutCount	0
ReplyToQ	Blanks
ReplyToQMgr	Blanks
GroupId	MQGI_NONE
MsgSeqNumber	1
Offset	0
MsgFlags	MQMF_NONE
OriginalLength	MQOL_UNDEFINED

- The message descriptor is then modified by the new message handle – any message descriptor fields explicitly set as properties in the new message handle take precedence over the message descriptor fields as described above.

The message properties are composed as follows:

- All properties from the original message handle which have MQCOPY_REPLY in the MQPD.CopyOptions
- All properties from the new message handle. For each property in the new message handle that has the same name as a property in the original message handle, the value is taken from the new message handle. The only exception to this rule is the special case when the property in the new

message handle has the same name as a property in the original message handle, but the value of the property is null. In this case the property is removed from the message.

The message data to be forwarded is taken from the MQPUT/MQPUT1 Buffer parameter.

MQACTP_REPORT

A report is being generated as a result of a previously retrieved message. The original message handle specifies the message causing the report to be generated.

The new message handle specifies any modifications to the properties (including any in the message descriptor) in the original message handle.

The message descriptor is composed as follows:

- If a MsgDesc is supplied on the MQPUT or MQPUT1 call, and MQPMO_MD_FOR_OUTPUT_ONLY is not in the MQPMO.Options, this is used as the message descriptor unmodified.
- If a MsgDesc is not supplied, or MQPMO_MD_FOR_OUTPUT_ONLY is in the MQPMO.Options then initial message descriptor fields are chosen as follows:

Table 59. Report message handle transformation

Field in MQMD	Value used
Report	If MQRO_PASS_DISCARD_AND_EXPIRY and MQRO_DISCARD_MSG are set: MQRO_DISCARD_MSG otherwise MQRO_NONE
MsgType	MQMT_REPORT
Expiry	If MQRO_PASS_DISCARD_AND_EXPIRY is set: Copied from the input message otherwise MQEI_UNLIMITED
MsgId	If MQPMO_NEW_MSG_ID is set: A new message identifier is generated else if MQRO_PASS_MSG_ID is set: Copied from the input message otherwise MQMI_NONE
CorrelId	If MQPMO_NEW_CORREL_ID is set: A new correlation identifier is generated else if MQRO_COPY_MSG_ID_TO_CORREL_ID is set: Copied from the MsgId field of the input message else if MQRO_PASS_CORREL_ID is set: Copied from the CorrelId field of the input message otherwise MQCI_NONE
BackoutCount	0
ReplyToQ	Blanks
ReplyToQMgr	Blanks
OriginalLength	Set to the <i>BufferLength</i>

- The message descriptor is then modified by the new message handle – any message descriptor fields explicitly set as properties in the new message handle take precedence over the message descriptor fields as described above.

The message properties are composed as follows:

- All properties from the original message handle which have MQCOPY_REPORT in the MQPD.CopyOptions
- All properties from the new message handle. For each property in the new message handle that has the same name as a property in the original message handle, the value is taken from the new message handle. The only exception to this rule is the special case when the property in the new message handle has the same name as a property in the original message handle, but the value of the property is null. In this case the property is removed from the message.

The Feedback field in the resultant MQMD represents the report that is to be generated. A Feedback value of MQFB_NONE causes the MQPUT or MQPUT1 call to fail with reason code MQRC_FEEDBACK_ERROR.

To choose the user data of the report message, WebSphere MQ consults the Report and Feedback fields in the resultant MQMD, and the Buffer and BufferLength parameters of the MQPUT or MQPUT1 call.

- If Feedback is MQFB_COA, MQFB_COD or MQFB_EXPIRATION then the value of Report is inspected.
- If any of the following cases is true, the full message data from Buffer for a length of BufferLength is used.
 - Feedback is MQFB_EXPIRATION and Report contains MQRO_EXPIRATION_WITH_FULL_DATA
 - Feedback is MQFB_COD and Report contains MQRO_COD_WITH_FULL_DATA
 - Feedback is MQFB_COA and Report contains MQRO_COA_WITH_FULL_DATA
- If any of the following cases is true, the first 100 bytes of the message (or BufferLength if this is less than 100) from Buffer are used
 - Feedback is MQFB_EXPIRATION and Report contains MQRO_EXPIRATION_WITH_DATA
 - Feedback is MQFB_COD and Report contains MQRO_COD_WITH_DATA
 - Feedback is MQFB_COA and Report contains MQRO_COA_WITH_DATA
- If Feedback is MQFB_EXPIRATION, MQFB_COD or MQFB_COA, and Report does not contain the *_WITH_FULL_DATA or *_WITH_DATA options relevant to that Feedback value, then no user data is included with the message.
- If Feedback takes a different value from those listed above, then Buffer and BufferLength are used as normal.

The derivation of the user data is shown in the following table:

Table 60. Source of user data

	MQFB_COA	MQFB_COD	MQFB_EXPIRATION
MQRO_EXPIRATION_WITH_FULL_DATA	None	None	Buffer(Bufferlength)

Table 60. Source of user data (continued)

	MQFB_COA	MQFB_COD	MQFB_EXPIRATION
MQRO_COD _WITH_FULL_DATA	None	Buffer(Bufferlength)	None
MQRO_COA _WITH_FULL_DATA	Buffer(Bufferlength)	None	None
MQRO_EXPIRATION _WITH_DATA	None	None	Buffer(First 100 bytes)
MQRO_COD _WITH_DATA	None	Buffer(First 100 bytes)	None
MQRO_COA _WITH_DATA	Buffer(First 100 bytes)	None	None

Context (MQHOBJ)

If MQPMO_PASS_IDENTITY_CONTEXT or MQPMO_PASS_ALL_CONTEXT is specified, this field must contain the input queue handle from which context information to be associated with the message being put is taken.

If neither MQPMO_PASS_IDENTITY_CONTEXT nor MQPMO_PASS_ALL_CONTEXT is specified, this field is ignored.

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

InvalidDestCount (MQLONG)

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 that is not in a distribution list.

Note: This field is set *only* if the *CompCode* parameter on the MQPUT or MQPUT1 call is MQCC_OK or MQCC_WARNING; it is *not* set if the *CompCode* parameter is MQCC_FAILED.

This is an output field. The initial value of this field is 0. This field is not set if *Version* is less than MQPMO_VERSION_2.

KnownDestCount (MQLONG)

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 that is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is not set if *Version* is less than MQPMO_VERSION_2.

NewMsgHandle (MQHMSG)

This is an optional handle to the message being put subject to the value of the *Action* field. It defines the properties of the message and overrides the values of the *OriginalMsgHandle*, if specified.

On return from the MQPUT or MQPUT1 call the contents of the handle will reflect the message that was actually put.

This is an input field. The initial value of this field is MQHM_NONE. This field is ignored if Version is less than MQPMO_VERSION_3.

Options (MQLONG)

Scope option.

You can specify any or none of the following options. If more than one option is required, the values you specify for the options can be used in the following ways:

- The values can be added together. Do not add the same constant more than once.
- The values can be combined using the bitwise OR operation, if the programming language supports bitwise operations.

Combinations that are not valid are noted; any other combinations are valid.

The following option controls the scope of the publications sent:

MQPMO_SCOPE_QMGR

The publication is sent only to subscribers that have subscribed on this queue manager. The publication is not forwarded to any remote publish/subscribe queue managers that have made a subscription to this queue manager. This overrides any behavior that has been set using the PUBSCOPE topic attribute.

Note: If not set, the publication scope is determined by the PUBSCOPE topic attribute.

Publishing options. The following options control the way messages are published to a topic:

MQPMO_SUPPRESS_REPLYTO

Any information specified in the *ReplyToQ* and *ReplyToQMGR* fields of the MQMD of this publication is not passed on to subscribers. If this option is used with a report option that requires a *ReplyToQ*, the call fails with MQRC_MISSING_REPLY_TO_Q.

MQPMO_RETAIN

The publication being sent is to be retained by the queue manager. This allows a subscriber to request a copy of this publication after the time it was published, by using the MQSUBRQ call. It also allows a publication to be sent to applications which make their subscription after the time this publication was made (unless they choose not to be sent it by using the option MQSO_NEW_PUBLICATIONS_ONLY). If an application is sent a publication which was retained, this is indicated by the MQIsRetained message property of that publication.

Only one publication can be retained at each node of the topic tree. Therefore, if there already is a retained publication for this topic, published by any other application, it is replaced with this publication. It is therefore better to avoid having more than one publisher retaining messages on the same topic.

When retained publications are requested by a subscriber, the subscription used might contain a wildcard in the topic, in which case a number of

retained publications might match (at various nodes in the topic tree) and several publications might be sent to the requesting application. See the description of the MQSUBRQ call for more details.

For information about how retained publications interact with subscription levels, see *WebSphere Publish/Subscribe User's Guide*.

If this option is used and the publication cannot be retained, the message is not published and the call fails with MQRC_PUT_NOT_RETAINED.

MQPMO_NOT_OWN_SUBS

Tells the queue manager that the application does not want to send any of its publications to subscriptions it owns. Subscriptions are considered to be owned by the same application if the connection handles are the same.

Syncpoint options. The following options relate to the participation of the MQPUT or MQPUT1 call within a unit of work:

MQPMO_SYNCPOINT

The request is to operate within the normal unit-of-work protocols. The message is not visible outside the unit of work until the unit of work is committed. If the unit of work is backed out, the message is deleted.

If neither this option nor MQPMO_NO_SYNCPOINT is specified, the inclusion of the put request in unit-of-work protocols is determined by the environment:

- On z/OS, the put request is within a unit of work.
- In all other environments, the put request is not within a unit of work.

Because of these differences, an application that you want to port must not allow this option to default; specify either MQPMO_SYNCPOINT or MQPMO_NO_SYNCPOINT explicitly.

Do not specify MQPMO_SYNCPOINT with MQPMO_NO_SYNCPOINT.

MQPMO_NO_SYNCPOINT

The request is to operate outside the normal unit-of-work protocols. The message is available immediately, and it cannot be deleted by backing out a unit of work.

If neither this option nor MQPMO_SYNCPOINT is specified, the inclusion of the put request in unit-of-work protocols is determined by the environment:

- On z/OS, the put request is within a unit of work.
- In all other environments, the put request is not within a unit of work.

Because of these differences, an application that you want to port must not allow this option to default; specify either MQPMO_SYNCPOINT or MQPMO_NO_SYNCPOINT explicitly.

Do not specify MQPMO_NO_SYNCPOINT with MQPMO_SYNCPOINT.

Message-identifier and correlation-identifier options. The following options request the queue manager to generate a new message identifier or correlation identifier:

MQPMO_NEW_MSG_ID

The queue manager replaces the contents of the *MsgId* field in MQMD with a new message identifier. This message identifier is sent with the message, and returned to the application on output from the MQPUT or MQPUT1 call.

This option can also be specified when the message is being put to a distribution list; see the description of the *MsgId* field in the MQPMR structure for details.

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

MQPMO_NEW_CORREL_ID

The queue manager replaces the contents of the *CorrelId* field in MQMD with a new correlation identifier. This correlation identifier is sent with the message, and returned to the application on output from the MQPUT or MQPUT1 call.

This option can also be specified when the message is being put to a distribution list; see the description of the *CorrelId* field in the MQPMR structure for details.

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

Group and segment options. The following options relate to the processing of messages in groups and segments of logical messages. Read the definitions that follow to help you to understand the option.

Physical message

This is the smallest unit of information that can be placed on or removed from a queue; it often corresponds to the information specified or retrieved on a single MQPUT, MQPUT1, or MQGET call. Every physical message has its own message descriptor (MQMD). Generally, physical messages are distinguished by differing values for the message identifier (*MsgId* field in MQMD), although this is not enforced by the queue manager.

Logical message

This is a single unit of application information. In the absence of system constraints, a logical message is the same as a physical message. But where logical messages are extremely large, system constraints might make it advisable or necessary to split a logical message into two or more physical messages, called *segments*.

A logical message that has been segmented consists of two or more physical messages that have the same nonnull group identifier (*GroupId* field in MQMD), and the same message sequence number (*MsgSeqNumber* field in MQMD). The segments are distinguished by differing values for the segment offset (*Offset* field in MQMD), which gives the offset of the data in the physical message from the start of the data in the logical message. Because each segment is a physical message, the segments in a logical message usually have differing message identifiers.

A logical message that has not been segmented, but for which segmentation has been permitted by the sending application, also has a nonnull group identifier, although in this case there is only one physical message with that group identifier if the logical message does not belong to a message group. Logical messages for which segmentation has been inhibited by the sending application have a null group identifier (MQGI_NONE), unless the logical message belongs to a message group.

Message group

This is a set of one or more logical messages that have the same nonnull group identifier. The logical messages in the group are distinguished by differing values for the message sequence number, which is an integer in

the range 1 through n, where n is the number of logical messages in the group. If one or more of the logical messages is segmented, there are more than n physical messages in the group.

MQPMO_LOGICAL_ORDER

This option tells the queue manager how the application puts messages in groups and segments of logical messages. It can be specified only on the MQPUT call; it is *not* valid on the MQPUT1 call.

If MQPMO_LOGICAL_ORDER is specified, it indicates that the application uses 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 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 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 puts 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, because the queue manager does this. Specifically, it means that the application does not need to set the *GroupId*, *MsgSeqNumber*, and *Offset* fields in MQMD, because the queue manager sets these to the appropriate values. The application needs to set only the *MsgFlags* field in MQMD, to indicate when messages belong to groups or are segments of logical messages, and to indicate the last message in a group or last segment of a logical message.

After a message group or logical message has been started, subsequent MQPUT calls must specify the appropriate MQMF_* flags in *MsgFlags* in MQMD. If the application tries to put a message that is not in a group when there is an unterminated message group, or put a message that is not a segment when there is an unterminated logical message, the call fails with reason code MQRC_INCOMPLETE_GROUP or MQRC_INCOMPLETE_MSG, as appropriate. However, the queue manager retains the information about the current message group or current logical message, and the application can terminate them by sending a message (possibly with no application message data) specifying MQMF_LAST_MSG_IN_GROUP or MQMF_LAST_SEGMENT as appropriate, before reissuing the MQPUT call to put the message that is not in the group or not a segment.

Table 61 on page 284 shows the combinations of options and flags that are valid, and the values of the *GroupId*, *MsgSeqNumber*, and *Offset* fields that the queue manager uses in each case. Combinations of options and flags that are not shown in the table are not valid. The columns in the table have the following meanings; Either means Yes or No:

LOG ORD

Whether the MQPMO_LOGICAL_ORDER option is specified on the call.

MIG Whether the MQMF_MSG_IN_GROUP or MQMF_LAST_MSG_IN_GROUP option is specified on the call.

SEG Whether the MQMF_SEGMENT or MQMF_LAST_SEGMENT option is specified on the call.

SEG OK

Whether the MQMF_SEGMENTATION_ALLOWED option is specified on the call.

Cur grp

Whether a current message group exists prior to the call.

Cur log msg

Whether a current logical message exists prior to the call.

Other columns

Show the values that the queue manager uses. Previous denotes the value used for the field in the previous message for the queue handle.

Table 61. MQPUT options relating to messages in groups and segments of logical messages

Options you specify				Group and log-msg status prior to call		Values the queue manager uses		
LOG ORD	MIG	SEG	SEG OK	Cur grp	Cur log msg	GroupId	MsgSeqNumber	Offset
Yes	No	No	No	No	No	MQGL_NONE	1	0
Yes	No	No	Yes	No	No	New group id	1	0
Yes	No	Yes	Either	No	No	New group id	1	0
Yes	No	Yes	Either	No	Yes	Previous group id	1	Previous offset + previous segment length
Yes	Yes	Either	Either	No	No	New group id	1	0
Yes	Yes	Either	Either	Yes	No	Previous group id	Previous sequence number + 1	0
Yes	Yes	Yes	Either	Yes	Yes	Previous group id	Previous sequence number	Previous offset + previous segment length
No	No	No	No	Either	Either	MQGL_NONE	1	0
No	No	No	Yes	Either	Either	New group id if MQGL_NONE, else value in field	1	0
No	No	Yes	Either	Either	Either	New group id if MQGL_NONE, else value in field	1	Value in field
No	Yes	No	Either	Either	Either	New group id if MQGL_NONE, else value in field	Value in field	0
No	Yes	Yes	Either	Either	Either	New group id if MQGL_NONE, else value in field	Value in field	Value in field

Table 61. MQPUT options relating to messages in groups and segments of logical messages (continued)

Options you specify	Group and log-msg status prior to call	Values the queue manager uses
<p>Notes:</p> <ul style="list-style-type: none"> MQPMO_LOGICAL_ORDER is not valid on the MQPUT1 call. For the <i>MsgId</i> field, the queue manager generates a new message identifier if MQPMO_NEW_MSG_ID or MQMI_NONE is specified, and uses the value in the field otherwise. For the <i>CorrelId</i> field, the queue manager generates a new correlation identifier if MQPMO_NEW_CORREL_ID is specified, and uses the value in the field otherwise. 		

When you specify MQPMO_LOGICAL_ORDER, the queue manager requires that all messages in a group and segments in a logical message are put with the same value in the *Persistence* field in MQMD, that is, all must be persistent, or all must be nonpersistent. If this condition is not satisfied, the MQPUT call fails with reason code MQRC_INCONSISTENT_PERSISTENCE.

The MQPMO_LOGICAL_ORDER option affects units of work as follows:

- If the first physical message in a group or logical message is put within a unit of work, all 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 that consists of many physical messages to be split across two or more consecutive units of work for the queue handle.
- If the first physical message in a group or logical message is *not* put within a unit of work, none of the other physical messages in the group or logical message can be put within a unit of work, if the same queue handle is used.

If these conditions are not satisfied, the MQPUT call fails with reason code MQRC_INCONSISTENT_UOW.

When MQPMO_LOGICAL_ORDER is specified, the MQMD supplied on the MQPUT call must not be less than MQMD_VERSION_2. If this condition is not satisfied, the call fails with reason code MQRC_WRONG_MD_VERSION.

If MQPMO_LOGICAL_ORDER is *not* specified, messages in groups and segments of logical messages can be put in any order, and it is not necessary to put complete message groups or complete logical messages. It is the application's responsibility to ensure that the *GroupId*, *MsgSeqNumber*, *Offset*, and *MsgFlags* fields have appropriate values.

Use this technique to restart a message group or logical message in the middle, after a system failure has occurred. When the system restarts, the application can set the *GroupId*, *MsgSeqNumber*, *Offset*, *MsgFlags*, and *Persistence* fields to the appropriate values, and then issue the MQPUT call with MQPMO_SYNCPOINT or MQPMO_NO_SYNCPOINT set as desired, but *without* specifying MQPMO_LOGICAL_ORDER. If this call is successful, the queue manager retains the group and segment information, and subsequent MQPUT calls using that queue handle can specify MQPMO_LOGICAL_ORDER as normal.

The group and segment information that the queue manager retains for the MQPUT call is separate from the group and segment information that it retains for the MQGET call.

For any given queue handle, the application can mix MQPUT calls that specify MQPMO_LOGICAL_ORDER with MQPUT calls that do not, but note the following points:

- If MQPMO_LOGICAL_ORDER is *not* specified, each successful MQPUT call causes the queue manager to set the group and segment information for the queue handle to the values specified by the application; this replaces the existing group and segment information retained by the queue manager for the queue handle.
- If MQPMO_LOGICAL_ORDER is *not* specified, the call does not fail if there is a current message group or logical message; the call might succeed with an MQCC_WARNING completion code. Table 62 shows the various cases that can arise. In these cases, if the completion code is not MQCC_OK, the reason code is one of the following (as appropriate):
 - MQRC_INCOMPLETE_GROUP
 - MQRC_INCOMPLETE_MSG
 - MQRC_INCONSISTENT_PERSISTENCE
 - MQRC_INCONSISTENT_UOW

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

Table 62. Outcome when MQPUT or MQCLOSE call is not consistent with group and segment information

Current call is	Previous call was MQPUT with MQPMO_LOGICAL_ORDER	Previous call was MQPUT without MQPMO_LOGICAL_ORDER
MQPUT with MQPMO_LOGICAL_ORDER	MQCC_FAILED	MQCC_FAILED
MQPUT without MQPMO_LOGICAL_ORDER	MQCC_WARNING	MQCC_OK
MQCLOSE with an unterminated group or logical message	MQCC_WARNING	MQCC_OK

For applications that put messages and segments in logical order, specify MQPMO_LOGICAL_ORDER, as this is the simplest option to use. This option relieves the application of the need to manage the group and segment information, because the queue manager manages that information. However, specialized applications might need more control than that provided by the MQPMO_LOGICAL_ORDER option, and this can be achieved by not specifying that option. If this is done, the application must ensure that the *GroupId*, *MsgSeqNumber*, *Offset*, and *MsgFlags* fields in MQMD are set correctly, before 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, must *not* specify MQPMO_LOGICAL_ORDER. There are two reasons for this:

- If the messages are retrieved and put in order, specifying MQPMO_LOGICAL_ORDER assigns a new group identifier to the

messages, and this might 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 might arrive out of order. By specifying neither MQPMO_LOGICAL_ORDER nor the corresponding MQGMO_LOGICAL_ORDER on the MQGET call, the forwarding application can retrieve and forward each physical message as soon as it arrives, without having to wait for the next one in logical order to arrive.

Applications that generate report messages for messages in groups or segments of logical messages must also not specify MQPMO_LOGICAL_ORDER when putting the report message.

MQPMO_LOGICAL_ORDER can be specified with any of the other MQPMO_* options.

Context options. The following options control the processing of message context:

MQPMO_NO_CONTEXT

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

MQPMO_DEFAULT_CONTEXT

The message is to have default context information associated with it, for both identity and origin. The queue manager sets the context fields in the message descriptor as follows:

Field in MQMD	Value used
<i>UserIdentifier</i>	Determined from the environment if possible; set to blanks otherwise.
<i>AccountingToken</i>	Determined from the environment if possible; set to MQACT_NONE otherwise.
<i>ApplIdentityData</i>	Set to blanks.
<i>PutApplType</i>	Determined from the environment.
<i>PutApplName</i>	Determined from the environment if possible; set to blanks otherwise.
<i>PutDate</i>	Set to the date when message is put.
<i>PutTime</i>	Set to the time when message is put.
<i>ApplOriginData</i>	Set to blanks.

For more information on message context, see the *WebSphere MQ Application Programming Guide*.

This is the default action if no context options are specified.

MQPMO_PASS_IDENTITY_CONTEXT

The message is to have context information associated with it. Identity context is taken from the queue handle specified in the *Context* field. Origin context information is generated by the queue manager in the same way that it is for MQPMO_DEFAULT_CONTEXT (see above for values). For more information on message context, see the *WebSphere MQ Application Programming Guide*.

For the MQPUT call, the queue must have been opened with the MQOO_PASS_IDENTITY_CONTEXT option (or an option that implies it). For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the MQOO_PASS_IDENTITY_CONTEXT option.

MQPMO_PASS_ALL_CONTEXT

The message is to have context information associated with it. Context is taken from the queue handle specified in the *Context* field. For more information on message context, see *WebSphere MQ Application Programming Guide*.

For the MQPUT call, the queue must have been opened with the MQOO_PASS_ALL_CONTEXT option (or an option that implies it). For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the MQOO_PASS_ALL_CONTEXT option.

MQPMO_SET_IDENTITY_CONTEXT

The message is to have context information associated with it. The application specifies the identity context in the MQMD structure. Origin context information is generated by the queue manager in the same way that it is for MQPMO_DEFAULT_CONTEXT (see above for values). For more information on message context, see the *WebSphere MQ Application Programming Guide*.

For the MQPUT call, the queue must have been opened with the MQOO_SET_IDENTITY_CONTEXT option (or an option that implies it). For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the MQOO_SET_IDENTITY_CONTEXT option.

MQPMO_SET_ALL_CONTEXT

The message is to have context information associated with it. The application specifies the identity, origin, and user context in the MQMD structure. For more information on message context, see the *WebSphere MQ Application Programming Guide*.

For the MQPUT call, the queue must have been opened with the MQOO_SET_ALL_CONTEXT option. For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the MQOO_SET_ALL_CONTEXT option.

You can specify only one of the MQPMO_*_CONTEXT context options. If you specify none, MQPMO_DEFAULT_CONTEXT is assumed.

Property options. The following option relates to the properties of the message:

MQPMO_MD_FOR_OUTPUT_ONLY

The message descriptor parameter should only be used for output to return the message descriptor of the message that was actually put. The message descriptor fields associated with the *NewMsgHandle* and/or *OriginalMsgHandle* fields of the MQPMO structure should be used for input.

If a valid message handle is not provided then the call fails with reason code MQRC_MD_ERROR.

Put response options. The following options control the response returned to an MQPUT or MQPUT1 call. You can specify only one of these options. If neither MQPMO_ASYNC_RESPONSE nor MQPMO_SYNC_RESPONSE is specified, MQPMO_RESPONSE_AS_Q_DEF or MQPMO_RESPONSE_AS_TOPIC_DEF is assumed.

MQPMO_ASYNC_RESPONSE

The MQPMO_ASYNC_RESPONSE option requests that an MQPUT or MQPUT1 operation is completed without the application waiting for the queue manager to complete the call. Using this option can improve messaging performance, particularly for applications using client bindings. An application can periodically check, using the MQSTAT verb, whether an error has occurred during any previous asynchronous calls.

With this option, only the following fields are guaranteed to be completed in the MQMD;

- ApplIdentityData
- PutAppType
- PutAppName
- ApplOriginData

Additionally, if either or both of MQPMO_NEW_MSG_ID or MQPMO_NEW_CORREL_ID are specified as options, the MsgId and CorrelId returned are also completed. (MQPMO_NEW_MSG_ID can be implicitly specified by specifying a blank MsgId field).

Only the fields specified above are completed. Other information that would normally be returned in the MQMD or MQPMO structure is undefined.

When requesting asynchronous put response for MQPUT or MQPUT1, a CompCode and Reason of MQCC_OK and MQRC_NONE does not necessarily mean that the message was successfully put to a queue. When developing an MQI application that uses asynchronous put response and requires confirmation that messages have been put to a queue you should check both CompCode & Reason codes from the put operations and also use MQSTAT to query asynchronous error information.

Although the success or failure of each individual MQPUT or MQPUT1 call may not be returned immediately, the first error that occurred under an asynchronous call can be determined later through a call to MQSTAT.

If a persistent message under syncpoint fails to be delivered using asynchronous put response, and you attempt to commit the transaction, the commit fails and the transaction is backed out with a completion code of MQCC_FAILED and a reason of MQRC_BACKED_OUT. The application can make a call to MQSTAT to determine the cause of a previous MQPUT or MQPUT1 failure.

MQPMO_SYNC_RESPONSE

Specifying this put response type ensures that the MQPUT or MQPUT1 operation is always issued synchronously. If the put operation is successful, all fields in the MQMD and MQPMO are completed.

This option is provided to ensure a synchronous response irrespective of the default put response value defined on the queue or topic object.

MQPMO_RESPONSE_AS_Q_DEF

If this value is specified for an MQPUT call, the put response type used is taken from the DEFPRESP value specified on the queue when it was first opened by the application. If a client application is connected to a queue manager at a level earlier than Version 7.0, it behaves as if MQPMO_SYNC_RESPONSE was specified.

If this option is specified for an MQPUT1 call, the DEFPRESP value from the queue definition is not used. If the MQPUT1 call is using

MQPMO_SYNCPOINT it behaves as for MQPMO_ASYNC_RESPONSE, and if it is using MQPMO_NO_SYNCPOINT it behaves as for MQPMO_SYNC_RESPONSE.

MQPMO_RESPONSE_AS_TOPIC_DEF

This is a synonym for MQPMO_RESPONSE_AS_Q_DEF for use with topic objects.

Other options. The following options control authorization checking, what happens when the queue manager is quiescing, and resolving queue and queue manager names:

MQPMO_ALTERNATE_USER_AUTHORITY

This indicates that the *AlternateUserId* field in the *ObjDesc* parameter of the MQPUT1 call contains a user identifier that is to be used to validate authority to put messages on the queue. The call can succeed only if this *AlternateUserId* is authorized to open the queue with the specified options, regardless of whether the user identifier under which the application is running is authorized to do so. (This does not apply to the context options specified, however, which are always checked against the user identifier under which the application is running.)

This option is valid only with the MQPUT1 call.

MQPMO_FAIL_IF QUIESCING

This option forces the MQPUT or MQPUT1 call to fail if the queue manager is in the quiescing state.

On z/OS, this option also forces the MQPUT or MQPUT1 call to fail if the connection (for a CICS or IMS application) is in the quiescing state.

The call returns completion code MQCC_FAILED with reason code MQRC_Q_MGR QUIESCING or MQRC_CONNECTION QUIESCING.

MQPMO_RESOLVE_LOCAL_Q

Use this option to fill *ResolvedQName* in the MQPMO structure with the name of the local queue to which the message is put, and *ResolvedQMgrName* with the name of the local queue manager that hosts the local queue. For more detail on this, see MQOO_RESOLVE_LOCAL_Q.

If you are authorized to put to a queue, you have the required authority to specify this flag on the MQPUT call; no special authority is needed.

Default option. If you need none of the options described, use the following option:

MQPMO_NONE

Use this value to indicate that no other options have been specified; all options assume their default values. MQPMO_NONE is defined to aid program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

This is an input field. The initial value of the *Options* field is MQPMO_NONE.

OriginalMsgHandle (MQHMSG)

This is an optional handle to a message. It may have been previously retrieved from a queue. The use of this handle is subject to the value of the *Action* field; see also NewMsgHandle.

The contents of the original message handle will not be altered by the MQPUT or MQPUT1 call.

This is an input field. The initial value of this field is MQHM_NONE. This field is ignored if Version is less than MQPMO_VERSION_3.

PubLevel (MQLONG)

The initial value of this field is 9. The level of subscription targeted by this publication. Only those subscriptions with the highest SubLevel less than or equal to this value will receive this publication. This value must be in the range zero to 9; zero is the lowest level.

For information, see *WebSphere Publish/Subscribe User's Guide*.

PutMsgRecFields (MQLONG)

This field contains flags that indicate which MQPMR fields are present in the put message records provided by the application. Use *PutMsgRecFields* only when the message is being put to a distribution list. The field is ignored if *RecsPresent* is zero, or both *PutMsgRecOffset* and *PutMsgRecPtr* are zero.

For fields that are present, the queue manager uses for each destination the values from the fields in the corresponding put message record. For fields that are absent, the queue manager uses the values from the MQMD structure.

Use one or more of the following flags to indicate which fields are present in the put message records:

MQPMRF_MSG_ID

Message-identifier field is present.

MQPMRF_CORREL_ID

Correlation-identifier field is present.

MQPMRF_GROUP_ID

Group-identifier field is present.

MQPMRF_FEEDBACK

Feedback field is present.

MQPMRF_ACCOUNTING_TOKEN

Accounting-token field is present.

If you specify this flag, specify either MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT in the *Options* field; if this condition is not satisfied, the call fails with reason code MQRC_PMO_RECORD_FLAGS_ERROR.

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

MQPMRF_NONE

No put-message record fields are present.

If this value is specified, either *RecsPresent* must be zero, or both *PutMsgRecOffset* and *PutMsgRecPtr* must be zero.

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

If *PutMsgRecFields* contains flags that are not valid, or put message records are provided but *PutMsgRecFields* has the value MQPMRF_NONE, the call fails with reason code MQRC_PMO_RECORD_FLAGS_ERROR.

This is an input field. The initial value of this field is MQPMRF_NONE. This field is ignored if *Version* is less than MQPMO_VERSION_2.

PutMsgRecOffset (MQLONG)

This is the offset in bytes of the first MQPMR put message record from the start of the MQPMO structure. The offset can be positive or negative. *PutMsgRecOffset* is used only when the message is being put to a distribution list. The field is ignored if *RecsPresent* is zero.

When the message is being put to a distribution list, an array of one or more MQPMR put message records can be provided in order to specify certain properties of the message for each destination individually; these properties are:

- Message identifier
- Correlation identifier
- Group identifier
- Feedback value
- Accounting token

You do not need to specify all these properties, but whatever subset you choose, specify the fields in the correct order. See the description of the MQPMR structure for further details.

Usually, there must 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 that 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.

The number of put message records can differ from the number of object records. If there are fewer put message records than object records, the message properties for the destinations that do not have put message records are taken from the corresponding fields in the message descriptor MQMD. If there are more put message records than object records, the excess are not used (although it must still be possible to access them). Put message records are optional, but if they are supplied there must be *RecsPresent* of them.

Provide the put message records in a similar way to the object records in MQOD, either by specifying an offset in *PutMsgRecOffset*, or by specifying an address in *PutMsgRecPtr*; for details of how to do this, see the *ObjectRecOffset* field described in “MQOD – Object descriptor” on page 249.

No more than one of *PutMsgRecOffset* and *PutMsgRecPtr* can be used; the call fails with reason code MQRC_PUT_MSG_RECORDS_ERROR if both are nonzero.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQPMO_VERSION_2.

PutMsgRecPtr (MQPTR)

This is the address of the first MQPMR put message record. Use *PutMsgRecPtr* only when the message is being put to a distribution list. The field is ignored if *RecsPresent* is zero.

You can use either *PutMsgRecPtr* or *PutMsgRecOffset* can be used to specify the put message records, but not both; see the description of the *PutMsgRecOffset* field above for details. If you do not use *PutMsgRecPtr*, set it to the null pointer or null bytes.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQPMO_VERSION_2.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

RecsPresent (MQLONG)

This is the number of MQPMR put message records or MQRR response records that have been provided by the application. This number can be greater than zero only if the message is being put to a distribution list. Put message records and response records are optional; the application need not provide any records, or it can choose to provide records of only one type. However, if the application provides records of both types, it must provide *RecsPresent* records of each type.

The value of *RecsPresent* need not be the same as the number of destinations in the distribution list. If too many records are provided, the excess are not used; if too few records are provided, default values are used for the message properties for those destinations that do not have put message records (see *PutMsgRecOffset* below).

If *RecsPresent* is less than zero, or is greater than zero but the message is not being put to a distribution list, the call fails with reason code MQRC_RECS_PRESENT_ERROR.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQPMO_VERSION_2.

ResolvedQMgrName (MQCHAR48)

This is the name of the destination queue manager after name resolution has been performed by the local queue manager. The name returned is the name of the queue manager that owns the queue identified by *ResolvedQName*, and can be the name of the local queue manager.

If *ResolvedQName* is a shared queue that is owned by the queue-sharing group to which the local queue manager belongs, *ResolvedQMgrName* is the name of the queue-sharing group. If the queue is owned by some other queue-sharing group, *ResolvedQName* can be the name of the queue-sharing group or the name of a queue manager that is a member of the queue-sharing group (the nature of the value returned is determined by the queue definitions that exist at the local queue manager).

A nonblank value is returned only if the object is a single queue; if the object is a distribution list or a topic, the value returned is undefined.

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

ResolvedQName (MQCHAR48)

This is the name of the destination queue after name resolution has been performed by the local queue manager. The name returned is the name of a queue that exists on the queue manager identified by *ResolvedQMgrName*.

A nonblank value is returned only if the object is a single queue; if the object is a distribution list or a topic, the value returned is undefined.

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

ResponseRecOffset (MQLONG)

This is the offset in bytes of the first MQRR response record from the start of the MQPMO structure. The offset can be positive or negative. *ResponseRecOffset* is used only when the message is being put to a distribution list. The field is ignored if *RecsPresent* is zero.

When putting the message to a distribution list, you can provide an array of one or more MQRR response records to identify the queues to which the message was not sent successfully (*CompCode* field in MQRR), and the reason for each failure (*Reason* field in MQRR). The message might not have been sent either because the queue failed to open, or because the put operation failed. The queue manager sets the response records only when the outcome of the call is mixed (that is, some messages were sent successfully while others failed, or all failed but for differing reasons); reason code `MQRC_MULTIPLE_REASONS` from the call indicates this case. If the same reason code applies to all queues, that reason is returned in the *Reason* parameter of the MQPUT or MQPUT1 call, and the response records are not set.

Usually, there are 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 that 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.

The number of response records can differ from the number of object records. If there are fewer response records than object records, the application might not be able to identify all the destinations for which the put operation failed, or the reasons for the failures. If there are more response records than object records, the excess are not used (although it must still be possible to access them). Response records are optional, but if they are supplied there must be *RecsPresent* of them.

Provide the response records in a similar way to the object records in MQOD, either by specifying an offset in *ResponseRecOffset*, or by specifying an address in

ResponseRecPtr; for details of how to do this, see the *ObjectRecOffset* field described in “MQOD – Object descriptor” on page 249. However, use no more than one of *ResponseRecOffset* and *ResponseRecPtr*; the call fails with reason code MQRC_RESPONSE_RECORDS_ERROR if both are nonzero.

For the MQPUT1 call, this field must be zero. This is because the response information (if requested) is returned in the response records specified by the object descriptor MQOD.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQPMO_VERSION_2.

ResponseRecPtr (MQPTR)

This is the address of the first MQRR response record. *ResponseRecPtr* is used only when the message is being put to a distribution list. The field is ignored if *RecsPresent* is zero.

Use either *ResponseRecPtr* or *ResponseRecOffset* to specify the response records, but not both; see the description of the *ResponseRecOffset* field above for details. If you do not use *ResponseRecPtr* set it to the null pointer or null bytes.

For the MQPUT1 call, this field must be the null pointer or null bytes. This is because the response information (if requested) is returned in the response records specified by the object descriptor MQOD.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQPMO_VERSION_2.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

StrucId (MQCHAR4)

This is the structure identifier; the value must be:

MQPMO_STRUC_ID

Identifier for put-message options structure.

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

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

Timeout (MQLONG)

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

UnknownDestCount (MQLONG)

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 that is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is not set if *Version* is less than MQPMO_VERSION_2.

Version (MQLONG)

Structure version number.

The value must be one of the following:

MQPMO_VERSION_1

Version-1 put-message options structure.

This version is supported in all environments.

MQPMO_VERSION_2

Version-2 put-message options structure.

This version is supported in the following environments: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

MQPMO_VERSION_3

Version-3 put-message options structure.

This version is supported in all environments.

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

MQPMO_CURRENT_VERSION

Current version of put-message options structure.

This is always an input field. The initial value of this field is MQPMO_VERSION_1.

Initial values and language declarations for MQPMO

Table 63. Initial values of fields in MQPMO

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQPMO_STRUC_ID	'PMOb'
<i>Version</i>	MQPMO_VERSION_1	1
<i>Options</i>	MQPMO_NONE	0
<i>Timeout</i>	None	-1
<i>Context</i>	None	0
<i>KnownDestCount</i>	None	0
<i>UnknownDestCount</i>	None	0
<i>InvalidDestCount</i>	None	0
<i>ResolvedQName</i>	None	Null string or blanks
<i>ResolvedQMGrName</i>	None	Null string or blanks
<i>RecsPresent</i>	None	0
<i>PutMsgRecFields</i>	MQPMRF_NONE	0

Table 63. Initial values of fields in MQPMO (continued)

Field name	Name of constant	Value of constant
<i>PutMsgRecOffset</i>	None	0
<i>ResponseRecOffset</i>	None	0
<i>PutMsgRecPtr</i>	None	Null pointer or null bytes
<i>ResponseRecPtr</i>	None	Null pointer or null bytes
<i>OriginalMsgHandle</i>	MQHM_NONE	0
<i>NewMsgHandle</i>	MQHM_NONE	0
<i>Action</i>	MQACTP_NEW	0
<i>PubLevel</i>	None	9

Notes:

1. The symbol `b` represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQPMO_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

```
MQPMO MyPMO = {MQPMO_DEFAULT};
```

C declaration

```
typedef struct tagMQPMO MQPMO;
struct tagMQPMO {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQLONG    Options;          /* Options that control the action of
                                MQPUT and MQPUT1 */
    MQLONG    Timeout;          /* Reserved */
    MQHOBJ    Context;          /* Object handle of input queue */
    MQLONG    KnownDestCount;   /* Number of messages sent
                                successfully to local queues */
    MQLONG    UnknownDestCount; /* Number of messages sent
                                successfully to remote queues */
    MQLONG    InvalidDestCount; /* Number of messages that could not
                                be sent */
    MQCHAR48  ResolvedQName;    /* Resolved name of destination
                                queue */
    MQCHAR48  ResolvedQMgrName; /* Resolved name of destination queue
                                manager */
    /* Ver:1 */
    MQLONG    RecsPresent;       /* Number of put message records or
                                response records present */
    MQLONG    PutMsgRecFields;   /* Flags indicating which MQPMR fields
                                are present */
    MQLONG    PutMsgRecOffset;   /* Offset of first put message record
                                from start of MQPMO */
    MQLONG    ResponseRecOffset; /* Offset of first response record
                                from start of MQPMO */
    MQPTR     PutMsgRecPtr;      /* Address of first put message
                                record */
    MQPTR     ResponseRecPtr;    /* Address of first response record */
    /* Ver:2 */
    MQHMSG    OriginalMsgHandle; /* Original message handle */
    MQHMSG    NewMsgHandle;      /* New message handle */
};
```

```

MQLONG    Action;           /* The action being performed */
MQLONG    PubLevel;        /* Subscription level */
/* Ver:3 */
};

```

COBOL declaration

```

** MQPMO structure
10 MQPMO.
** Structure identifier
15 MQPMO-STRUCID          PIC X(4).
** Structure version number
15 MQPMO-VERSION         PIC S9(9) BINARY.
** Options that control the action of MQPUT and MQPUT1
15 MQPMO-OPTIONS         PIC S9(9) BINARY.
** Reserved
15 MQPMO-TIMEOUT         PIC S9(9) BINARY.
** Object handle of input queue
15 MQPMO-CONTEXT         PIC S9(9) BINARY.
** Number of messages sent successfully to local queues
15 MQPMO-KNOWNDSTCOUNT  PIC S9(9) BINARY.
** Number of messages sent successfully to remote queues
15 MQPMO-UNKNOWNDSTCOUNT PIC S9(9) BINARY.
** Number of messages that could not be sent
15 MQPMO-INVALIDDSTCOUNT PIC S9(9) BINARY.
** Resolved name of destination queue
15 MQPMO-RESOLVEDQNAME   PIC X(48).
** Resolved name of destination queue manager
15 MQPMO-RESOLVEDQMGRNAME PIC X(48).
** Number of put message records or response records present
15 MQPMO-RECSPRESENT     PIC S9(9) BINARY.
** Flags indicating which MQPMR fields are present
15 MQPMO-PUTMSGRECFIELDS PIC S9(9) BINARY.
** Offset of first put message record from start of MQPMO
15 MQPMO-PUTMSGRECOFFSET PIC S9(9) BINARY.
** Offset of first response record from start of MQPMO
15 MQPMO-RESPONSERECOFFSET PIC S9(9) BINARY.
** Address of first put message record
15 MQPMO-PUTMSGRECPTR    POINTER.
** Address of first response record
15 MQPMO-RESPONSERECPTR  POINTER.
** Original message handle
15 MQPMO-ORIGINALMSGHANDLE PIC S9(18) BINARY.
** New message handle
15 MQPMO-NEWMSGHANDLE    PIC S9(18) BINARY.
** The action being performed
15 MQPMO-ACTION          PIC S9(9) BINARY.
** Publish level
15 MQPMO-PUBLEVEL       PIC S9(9) BINARY.

```

PL/I declaration

```

dcl
1 MQPMO based,
3 StrucId          char(4),          /* Structure identifier */
3 Version          fixed bin(31), /* Structure version number */
3 Options          fixed bin(31), /* Options that control the action
                                of MQPUT and MQPUT1 */
3 Timeout         fixed bin(31), /* Reserved */
3 Context         fixed bin(31), /* Object handle of input queue */
3 KnownDestCount  fixed bin(31), /* Number of messages sent
                                successfully to local queues */
3 UnknownDestCount fixed bin(31), /* Number of messages sent
                                successfully to remote queues */
3 InvalidDestCount fixed bin(31), /* Number of messages that could
                                not be sent */
3 ResolvedQName   char(48),        /* Resolved name of destination
                                queue */

```

3	ResolvedQMgrName	char(48),	/* Resolved name of destination queue manager */
3	RecsPresent	fixed bin(31),	/* Number of put message records or response records present */
3	PutMsgRecFields	fixed bin(31),	/* Flags indicating which MQPMR fields are present */
3	PutMsgRecOffset	fixed bin(31),	/* Offset of first put message record from start of MQPMO */
3	ResponseRecOffset	fixed bin(31),	/* Offset of first response record from start of MQPMO */
3	PutMsgRecPtr	pointer,	/* Address of first put message record */
3	ResponseRecPtr	pointer,	/* Address of first response record */
3	OriginalMsgHandle	fixed bin(63),	/* Original message handle */
3	NewMsgHandle	fixed bin(63);	/* New message handle */
3	Action	fixed bin(31);	/* The action being performed */
3	PubLevel	fixed bin(31);	/* Publish level */

System/390 assembler declaration

MQPMO	DSECT		
MQPMO_STRUCID	DS	CL4	Structure identifier
MQPMO_VERSION	DS	F	Structure version number
MQPMO_OPTIONS	DS	F	Options that control the action of MQPUT and MQPUT1
*			
MQPMO_TIMEOUT	DS	F	Reserved
MQPMO_CONTEXT	DS	F	Object handle of input queue
MQPMO_KNOWNDSTCOUNT	DS	F	Number of messages sent successfully to local queues
*			
MQPMO_UNKNOWNDSTCOUNT	DS	F	Number of messages sent successfully to remote queues
*			
MQPMO_INVALIDDSTCOUNT	DS	F	Number of messages that could not be sent
*			
MQPMO_RESOLVEDQNAME	DS	CL48	Resolved name of destination queue
MQPMO_RESOLVEDQMGRNAME	DS	CL48	Resolved name of destination queue manager
*			
MQPMO_RECSPRESENT	DS	F	Number of put message records or response records present
*			
MQPMO_PUTMSGRECFIELDS	DS	F	Flags indicating which MQPMR fields are present
*			
MQPMO_PUTMSGRECOFFSET	DS	F	Offset of first put message record from start of MQPMO
*			
MQPMO_RESPONSERECOFFSET	DS	F	Offset of first response record from start of MQPMO
*			
MQPMO_PUTMSGRECPtr	DS	F	Address of first put message record
*			
MQPMO_RESPONSERECPtr	DS	F	Address of first response record
MQPMO_ORIGINALMSGHANDLE	DS	D	Original message handle
MQPMO_NEWMSGHANDLE	DS	D	New message handle
MQPMO_ACTION	DS	F	The action being performed
MQPMO_PUBLEVEL	DS	F	Publish level
*			
MQPMO_LENGTH	EQU	*-MQPMO	
	ORG	MQPMO	
MQPMO_AREA	DS	CL(MQPMO_LENGTH)	

Visual Basic declaration

Type	MQPMO	
StrucId	As String*4	'Structure identifier'
Version	As Long	'Structure version number'
Options	As Long	'Options that control the action of 'MQPUT and MQPUT1'
Timeout	As Long	'Reserved'
Context	As Long	'Object handle of input queue'
KnownDestCount	As Long	'Number of messages sent successfully 'to local queues'

UnknownDestCount	As Long	'Number of messages sent successfully' 'to remote queues'
InvalidDestCount	As Long	'Number of messages that could not be' 'sent'
ResolvedQName	As String*48	'Resolved name of destination queue'
ResolvedQMgrName	As String*48	'Resolved name of destination queue' 'manager'
RecsPresent	As Long	'Number of put message records or' 'response records present'
PutMsgRecFields	As Long	'Flags indicating which MQPMP fields' 'are present'
PutMsgRecOffset	As Long	'Offset of first put message record' 'from start of MQPMO'
ResponseRecOffset	As Long	'Offset of first response record from' 'start of MQPMO'
PutMsgRecPtr	As MQPTR	'Address of first put message record'
ResponseRecPtr	As MQPTR	'Address of first response record'
End Type		

MQPMR – Put-message record

The following table summarizes the fields in the structure.

Table 64. Fields in MQPMR

Field	Description	Topic
<i>MsgId</i>	Message identifier	MsgId
<i>CorrelId</i>	Correlation identifier	CorrelId
<i>GroupId</i>	Group identifier	GroupId
<i>Feedback</i>	Feedback or reason code	Feedback
<i>AccountingToken</i>	Accounting token	AccountingToken

Overview for MQPMR

Availability: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Purpose: Use the MQPMR structure to specify various message properties for a single destination when putting a message to a distribution list. MQPMR is an input/output structure for the MQPUT and MQPUT1 calls.

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

Usage: By providing an array of these structures on the MQPUT or MQPUT1 call, you can specify different values for each destination queue in a distribution list. Some of the fields are input only, others are input/output.

Note: This structure is unusual in that it does not have a fixed layout. The fields in this structure are optional, and the presence or absence of each field is indicated by the flags in the *PutMsgRecFields* field in MQPMO. Fields that are present *must occur in the following order:*

- *MsgId*
- *CorrelId*

- *GroupId*
- *Feedback*
- *AccountingToken*

Fields that are absent occupy no space in the record.

Because MQPMR does not have a fixed layout, no definition of it is provided in the header, COPY, and INCLUDE files for the supported programming languages. The application programmer must create a declaration containing the fields that are required by the application, and set the flags in *PutMsgRecFields* to indicate the fields that are present.

Fields for MQPMR

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

AccountingToken (MQBYTE32)

This is the accounting token to be used for the message sent to the queue whose name was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *AccountingToken* field in MQMD for a put to a single queue. See the description of *AccountingToken* in “MQMD – Message descriptor” on page 181 for information about the content of this field.

If this field is not present, the value in MQMD is used.

This is an input field.

CorrelId (MQBYTE24)

This is the correlation identifier to be used for the message sent to the queue whose name was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *CorrelId* field in MQMD for a put to a single queue.

If this field is not present in the MQPMR record, or there are fewer MQPMR records than destinations, the value in MQMD is used for those destinations that do not have an MQPMR record containing a *CorrelId* field.

If MQPMO_NEW_CORREL_ID is specified, a *single* new correlation identifier is generated and used for all the destinations in the distribution list, regardless of whether they have MQPMR records. This is different from the way that MQPMO_NEW_MSG_ID is processed (see *MsgId* field).

This is an input/output field.

Feedback (MQLONG)

This is the feedback code to be used for the message sent to the queue whose name was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *Feedback* field in MQMD for a put to a single queue.

If this field is not present, the value in MQMD is used.

This is an input field.

GroupId (MQBYTE24)

This is the group identifier to be used for the message sent to the queue whose name was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *GroupId* field in MQMD for a put to a single queue.

If this field is not present in the MQPMR record, or there are fewer MQPMR records than destinations, the value in MQMD is used for those destinations that do not have an MQPMR record containing a *GroupId* field. The value is processed as documented in Table 61 on page 284, but with the following differences:

- In those cases where a new group identifier would be used, the queue manager generates a different group identifier for each destination (that is, no two destinations have the same group identifier).
- In those cases where the value in the field would be used, the call fails with reason code MQRC_GROUP_ID_ERROR.

This is an input/output field.

MsgId (MQBYTE24)

This is the message identifier to be used for the message sent to the queue whose name was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *MsgId* field in MQMD for a put to a single queue.

If this field is not present in the MQPMR record, or there are fewer MQPMR records than destinations, the value in MQMD is used for those destinations that do not have an MQPMR record containing a *MsgId* field. If that value is MQMI_NONE, a new message identifier is generated for *each* of those destinations (that is, no two of those destinations have the same message identifier).

If MQPMO_NEW_MSG_ID is specified, new message identifiers are generated for all the destinations in the distribution list, regardless of whether they have MQPMR records. This is different from the way that MQPMO_NEW_CORREL_ID is processed (see *CorrelId* field).

This is an input/output field.

Initial values and language declarations for MQPMR

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 to declare the structure if all the fields are required.

C declaration

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

COBOL declaration

```
** MQPMR structure
  10 MQPMR.
**   Message identifier
    15 MQPMR-MSGID      PIC X(24).
**   Correlation identifier
    15 MQPMR-CORRELID  PIC X(24).
**   Group identifier
    15 MQPMR-GROUPID   PIC X(24).
**   Feedback or reason code
    15 MQPMR-FEEDBACK  PIC S9(9) BINARY.
**   Accounting token
    15 MQPMR-ACCOUNTINGTOKEN PIC X(32).
```

PL/I declaration

```
dcl
  1 MQPMR based,
    3 MsgId      char(24),      /* Message identifier */
    3 CorrelId   char(24),      /* Correlation identifier */
    3 GroupId    char(24),      /* Group identifier */
    3 Feedback   fixed bin(31), /* Feedback or reason code */
    3 AccountingToken char(32); /* Accounting token */
```

Visual Basic declaration

```
Type MQPMR
  MsgId      As MQBYTE24 'Message identifier'
  CorrelId   As MQBYTE24 'Correlation identifier'
  GroupId    As MQBYTE24 'Group identifier'
  Feedback   As Long      'Feedback or reason code'
  AccountingToken As MQBYTE32 'Accounting token'
End Type
```

MQRFH – Rules and formatting header

Overview for MQRFH

Availability: All WebSphere MQ systems, plus WebSphere MQ clients connected to these systems.

Purpose: The MQRFH structure defines the layout of the rules and formatting header. Use this header to send string data in the form of name/value pairs.

Format name: MQFMT_RF_HEADER.

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

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

Fields for MQRFH

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

CodedCharSetId (MQLONG)

This specifies the character set identifier of the data that follows *NameValueString*; it does not apply to character data in the MQRFH structure itself.

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

MQCCSI_INHERIT

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

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

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

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG)

This specifies the numeric encoding of the data that follows *NameValueString*; it does not apply to numeric data in the MQRFH structure itself.

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

The initial value of this field is MQENC_NATIVE.

Flags (MQLONG)

The following can be specified:

MQRFH_NONE

No flags.

The initial value of this field is MQRFH_NONE.

Format (MQCHAR8)

This specifies the format name of the data that follows *NameValueString*.

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

The initial value of this field is MQFMT_NONE.

NameValueString (MQCHARn)

This is a variable-length character string containing name/value pairs in the form:
name1 value1 name2 value2 name3 value3 ...

Each name or value must be separated from the adjacent name or value by one or more blank characters; these blanks are not significant. A name or value can contain significant blanks by prefixing and suffixing the name or value with the double-quote character; all characters between the open double-quote and the

matching close double-quote are treated as significant. In the following example, the name is FAMOUS_WORDS, and the value is Hello World:

```
FAMOUS_WORDS "Hello World"
```

A name or value can contain any characters other than the null character (which acts as a delimiter for *NameValueString*; see below). However, to assist interoperability an application can restrict names to the following characters:

- First character: upper or lowercase alphabetic (A through Z, or a through z), or underscore.
- Subsequent characters: upper or lowercase alphabetic, decimal digit (0 through 9), underscore, hyphen, or dot.

If a name or value contains one or more double-quote characters, the name or value must be enclosed in double quotes, and each double quote within the string must be doubled:

```
Famous_Words "The program displayed ""Hello World"""
```

Names and values are case sensitive, that is, lowercase letters are not considered to be the same as uppercase letters. For example, FAMOUS_WORDS and Famous_Words are two different names.

The length in bytes of *NameValueString* is equal to *StrucLength* minus MQRFH_STRUC_LENGTH_FIXED. To avoid problems converting the user data in some environments, make this length a multiple of four. Pad *NameValueString* with blanks to this length, or terminate it earlier by placing a null character following the last significant character in the string. The null character and the bytes following it, up to the specified length of *NameValueString*, are ignored.

Note: Because the length of this field is not fixed, the field is omitted from the declarations of the structure that are provided for the supported programming languages.

StrucId (MQCHAR4)

This is the structure identifier; the value must be:

```
MQRFH_STRUC_ID
```

Identifier for rules and formatting header structure.

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

The initial value of this field is MQRFH_STRUC_ID.

StrucLength (MQLONG)

This is the length in bytes of the MQRFH structure, including the *NameValueString* field at the end of the structure. The length does *not* include any user data that follows the *NameValueString* field.

To avoid problems converting the user data in some environments, *StrucLength* must be a multiple of four.

The following constant gives the length of the *fixed* part of the structure, that is, the length excluding the *NameValueString* field:

MQRFH_STRUC_LENGTH_FIXED

Length of fixed part of MQRFH structure.

The initial value of this field is MQRFH_STRUC_LENGTH_FIXED.

Version (MQLONG)

This is the structure version number; the value must be:

MQRFH_VERSION_1

Version-1 rules and formatting header structure.

The initial value of this field is MQRFH_VERSION_1.

Initial values and language declarations for MQRFH

Table 65. Initial values of fields in MQRFH for MQRFH

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

Notes:

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

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

C declaration

```
typedef struct tagMQRFH MQRFH;
struct tagMQRFH {
    MQCHAR4  StrucId;          /* Structure identifier */
    MQLONG   Version;         /* Structure version number */
    MQLONG   StrucLength;     /* Total length of MQRFH including
                               NameValueString */
    MQLONG   Encoding;       /* Numeric encoding of data that follows
                               NameValueString */
    MQLONG   CodedCharSetId; /* Character set identifier of data that
                               follows NameValueString */
    MQCHAR8  Format;         /* Format name of data that follows
                               NameValueString */
    MQLONG   Flags;         /* Flags */
};
```

COBOL declaration

```
** MQRFH structure
10 MQRFH.
** Structure identifier
15 MQRFH-STRUCID PIC X(4).
```

```

**   Structure version number
15 MQRFH-VERSION      PIC S9(9) BINARY.
**   Total length of MQRFH including NAMEVALUESTRING
15 MQRFH-STRUCLNGTH  PIC S9(9) BINARY.
**   Numeric encoding of data that follows NAMEVALUESTRING
15 MQRFH-ENCODING    PIC S9(9) BINARY.
**   Character set identifier of data that follows NAMEVALUESTRING
15 MQRFH-CODEDCHARSETID PIC S9(9) BINARY.
**   Format name of data that follows NAMEVALUESTRING
15 MQRFH-FORMAT      PIC X(8).
**   Flags
15 MQRFH-FLAGS       PIC S9(9) BINARY.

```

PL/I declaration

```

dcl
1 MQRFH based,
3 StrucId      char(4),      /* Structure identifier */
3 Version      fixed bin(31), /* Structure version number */
3 StrucLength  fixed bin(31), /* Total length of MQRFH including
                             NameValueString */
3 Encoding     fixed bin(31), /* Numeric encoding of data that
                             follows NameValueString */
3 CodedCharSetId fixed bin(31), /* Character set identifier of data that
                             follows NameValueString */
3 Format        char(8),      /* Format name of data that follows
                             NameValueString */
3 Flags        fixed bin(31); /* Flags */

```

System/390 assembler declaration

```

MQRFH          DSECT
MQRFH_STRUCID  DS   CL4  Structure identifier
MQRFH_VERSION  DS   F    Structure version number
MQRFH_STRUCLNGTH DS   F    Total length of MQRFH including
*              NAMEVALUESTRING
MQRFH_ENCODING DS   F    Numeric encoding of data that follows
*              NAMEVALUESTRING
MQRFH_CODEDCHARSETID DS   F    Character set identifier of data that
*              follows NAMEVALUESTRING
MQRFH_FORMAT   DS   CL8  Format name of data that follows
*              NAMEVALUESTRING
MQRFH_FLAGS    DS   F    Flags
*
MQRFH_LENGTH   EQU   *-MQRFH
                ORG   MQRFH
MQRFH_AREA     DS   CL(MQRFH_LENGTH)

```

Visual Basic declaration

```

Type MQRFH
  StrucId      As String*4 'Structure identifier'
  Version      As Long     'Structure version number'
  StrucLength  As Long     'Total length of MQRFH including
                          'NameValueString'
  Encoding     As Long     'Numeric encoding of data that follows'
                          'NameValueString'
  CodedCharSetId As Long   'Character set identifier of data that'
                          'follows NameValueString'
  Format        As String*8 'Format name of data that follows'
                          'NameValueString'
  Flags        As Long     'Flags'
End Type

```

MQRFH2 – Rules and formatting header 2

Overview for MQRFH2

Availability: All WebSphere MQ systems, plus WebSphere MQ clients connected to these systems.

Purpose: The MQRFH2 header is based on the MQRFH header, but it allows Unicode strings to be transported without translation, and it can carry numeric datatypes.

The MQRFH2 structure defines the format of the version-2 rules and formatting header. Use this header to send data that has been encoded using an XML-like syntax. A message can contain two or more MQRFH2 structures in series, with user data optionally following the last MQRFH2 structure in the series.

Format name: MQFMT_RF_HEADER_2.

Character set and encoding: Special rules apply to the character set and encoding used for the MQRFH2 structure:

- Fields other than *NameValueData* are in the character set and encoding given by the *CodedCharSetId* and *Encoding* fields in the header structure that precedes MQRFH2, or by those fields in the MQMD structure if the MQRFH2 is at the start of the application message data.

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

When MQGMO_CONVERT is specified on the MQGET call, the queue manager converts these fields to the requested character set and encoding.

- *NameValueData* is in the character set given by the *NameValueCCSID* field. Only certain Unicode character sets are valid for *NameValueCCSID* (see the description of *NameValueCCSID* for details).

Some character sets have a representation that depends on the encoding. If *NameValueCCSID* is one of these character sets, *NameValueData* must be in the same encoding as the other fields in the MQRFH2.

When MQGMO_CONVERT is specified on the MQGET call, the queue manager converts *NameValueData* to the requested encoding, but does not change its character set.

Fields for MQRFH2

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

CodedCharSetId (MQLONG)

This specifies the character set identifier of the data that follows the last *NameValueData* field; it does not apply to character data in the MQRFH2 structure itself.

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

MQCCSI_INHERIT

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

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

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

The initial value of this field is MQCCSI_INHERIT.

Encoding (MQLONG)

This specifies the numeric encoding of the data that follows the last *NameValueData* field; it does not apply to numeric data in the MQRFH2 structure itself.

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

The initial value of this field is MQENC_NATIVE.

Flags (MQLONG)

The following value must be specified:

MQRFH_NONE

No flags.

The initial value of this field is MQRFH_NONE.

Format (MQCHAR8)

This specifies the format name of the data that follows the last *NameValueData* field.

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

The initial value of this field is MQFMT_NONE.

NameValueCCSID (MQLONG)

This specifies the coded character set identifier of the data in the *NameValueData* field. This is different from the character set of the other strings in the MQRFH2 structure, and can be different from the character set of the data (if any) that follows the last *NameValueData* field at the end of the structure.

NameValueCCSID must have one of the following values:

CCSID	Meaning
1200	UCS-2 open-ended
13488	UCS-2 2.0 subset
17584	UCS-2 2.1 subset (includes the Euro symbol)
1208	UTF-8

For the UCS-2 character sets, the encoding (byte order) of the *NameValueData* must be the same as the encoding of the other fields in the MQRFH2 structure. Surrogate characters (X'D800' through X'DFFF') are not supported.

Note: If *NameValueCCSID* does not have one of the values listed above, and the MQRFH2 structure requires conversion on the MQGET call, the call completes with reason code MQRC_SOURCE_CCSSID_ERROR and the message is returned unconverted.

The initial value of this field is 1208.

NameValueData (MQCHARn)

This is a variable-length character string containing data encoded using an XML-like syntax. The length in bytes of this string is given by the *NameValueLength* field that precedes the *NameValueData* field; this length must be a multiple of four.

The *NameValueLength* and *NameValueData* fields are optional, but if present they must occur as a pair and be adjacent. The pair of fields can be repeated as many times as required, for example:

```
length1 data1 length2 data2 length3 data3
```

Note:

1. Because these fields are optional, they are omitted from the declarations of the structure that are provided for the various programming languages supported.
2. For further information on the method of terminating the following *NameValue* fields, see topic *NameValueString*

NameValueData is *not* converted to the character set specified on the MQGET call when the message is retrieved with the MQGMO_CONVERT option in effect; *NameValueData* remains in its original character set. However, *NameValueData* is converted to the encoding specified on the MQGET call.

Syntax of name/value data: The string consists of a single *folder* that contains zero or more properties. The folder is delimited by XML start and end tags whose name is the name of the folder:

```
<folder> property1 property2 ... </folder>
```

Optionally, the content='properties' element can be included in the folder start tag. This indicates that the content of the folder is to be treated as message properties. This element must only be used with user-defined folders and not IBM-defined folders, for example, <wmq> or <jms>.

For example:

```
<com.ourcompany content='properties'> ... </com.ourcompany>
```

Characters following the folder end tag, up to the length defined by *NameValueLength*, must be blank. Within the folder, each property is composed of a name and a value, and optionally a data type:

```
<name dt="datatype">value</name>
```

In these examples:

- Specify the delimiter characters (<, =, ", /, and >) exactly as shown. The parameter value can be wrapped in either double quotes or apostrophes, for example dt="datatype" or dt='datatype'
- name is the user-specified name of the property; see below for more information about names.

- `datatype` is an optional user-specified data type of the property; see below for valid data types.
- `value` is the user-specified value of the property; see below for more information about values.
- Blanks are significant between the `>` character that precedes a value, and the `<` character that follows the value, and at least one blank must precede `dt=`. Elsewhere you can code blanks freely between tags, or preceding or following tags (for example, in order to improve readability); these blanks are not significant.

If properties are related to each other, you can group them together by enclosing them within XML start and end tags whose name is the name of the group:

```
<folder> <group> property1 property2 ... </group> </folder>
```

Groups can be nested within other groups, without limit, and a given group can occur more than once within a folder. A folder can also contain some properties in groups and other properties not in groups.

Names of properties, groups, and folders: The names of properties, groups, and folders must be valid XML tag names, with the exception of the colon character, which is not permitted in a property, group, or folder name. In particular:

- Names must start with a letter or an underscore. Valid letters are defined in the W3C XML specification, and consist essentially of Unicode categories Ll, Lu, Lo, Lt, and Nl.
- The remaining characters in a name can be letters, decimal digits, underscores, hyphens, or dots. These correspond to Unicode categories Ll, Lu, Lo, Lt, Nl, Mc, Mn, Lm, and Nd.
- The Unicode compatibility characters (X'F900' and above) are not permitted in any part of a name.
- Names must not start with the string XML in any mixture of upper or lowercase.

In addition:

- Names are case-sensitive. For example, ABC, abc, and Abc are three different names.
- Each folder has a separate name space. As a result, a group or property in one folder does not conflict with a group or property of the same name in another folder.
- Groups and properties occupy the same name space within a folder. As a result, a property cannot have the same name as a group within the folder containing that property.

Generally, programs that analyze the *NameValueData* field must ignore properties or groups that have names that the program does not recognize, provided that those properties or groups are correctly formed.

Data types of properties: Each property can have an optional data type. If specified, the data type must be one of the following values, in upper, lower, or mixed case:

Data type	Used for
string	Any sequence of characters. Certain characters must be specified using escape sequences (see below).
boolean	The character 0 or 1 (1 denotes TRUE).
bin.hex	Hexadecimal digits representing octets.

Data type	Used for
i1	Integer number in the range -128 through +127, expressed using only decimal digits and optional sign.
i2	Integer number in the range -32 768 through +32 767, expressed using only decimal digits and optional sign.
i4	Integer number in the range -2 147 483 648 through +2 147 483 647, expressed using only decimal digits and optional sign.
i8	Integer number in the range -9 223 372 036 854 775 808 through +9 223 372 036 854 775 807, expressed using only decimal digits and optional sign.
int	Integer number in the range -9 223 372 036 854 775 808 through +9 223 372 036 854 775 807, expressed using only decimal digits and optional sign. This can be used in place of i1, i2, i4, or i8 if the sender does not wish to imply a particular precision.
r4	Floating-point number with magnitude in the range 1.175E-37 through 3.402 823 47E+38, expressed using decimal digits, optional sign, optional fractional digits, and optional exponent.
r8	Floating-point number with magnitude in the range 2.225E-307 through 1.797 693 134 862 3E+308 expressed using decimal digits, optional sign, optional fractional digits, and optional exponent.

Values of properties: The value of a property can consist of any characters, except as detailed below. Each occurrence in the value of a character marked as *mandatory* must be replaced by the corresponding escape sequence. Each occurrence in the value of a character marked as *optional* can be replaced by the corresponding escape sequence, but this is not required.

Character	Escape sequence	Usage
&	&	Mandatory
<	<	Mandatory
>	>	Optional
"	"	Optional
'	'	Optional

Note: The & character at the start of an escape sequence must *not* be replaced by &.

In the following example, the blanks in the value are significant; however, no escape sequences are needed:

```
<Famous_Words>The program displayed "Hello World"</Famous_Words>
```

NameValueLength (MQLONG)

This specifies the length in bytes of the data in the *NameValueData* field. To avoid problems with data conversion of the data (if any) that *follows* the *NameValueData* field, *NameValueLength* must be a multiple of four.

Note: The *NameValueLength* and *NameValueData* fields are optional, but if present they must occur as a pair and be adjacent. The pair of fields can be repeated as many times as required, for example:

```
length1 data1 length2 data2 length3 data3
```

Because these fields are optional, they are omitted from the declarations of the structure that are provided for the various programming languages supported.

StrucId (MQCHAR4)

This is the structure identifier; the value must be:

MQRFH_STRUC_ID

Identifier for rules and formatting header structure.

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

The initial value of this field is `MQRFH_STRUC_ID`.

StrucLength (MQLONG)

This is the length in bytes of the `MQRFH2` structure, including the *NameValueLength* and *NameValueData* fields at the end of the structure. It is valid for there to be multiple pairs of *NameValueLength* and *NameValueData* fields at the end of the structure, in the sequence:

`length1, data1, length2, data2, ...`

StrucLength does *not* include any user data that may follow the last *NameValueData* field at the end of the structure.

To avoid problems with converting the user data in some environments, *StrucLength* must be a multiple of four.

The following constant gives the length of the *fixed* part of the structure, that is, the length excluding the *NameValueLength* and *NameValueData* fields:

MQRFH_STRUC_LENGTH_FIXED_2

Length of fixed part of `MQRFH2` structure.

The initial value of this field is `MQRFH_STRUC_LENGTH_FIXED_2`.

Version (MQLONG)

This is the structure version number; the value must be:

MQRFH_VERSION_2

Version-2 rules and formatting header structure.

The initial value of this field is `MQRFH_VERSION_2`.

Initial values and language declarations for MQRFH2

Table 66. Initial values of fields in `MQRFH2` for `MQRFH2`

Field name	Name of constant	Value of constant
<i>StrucId</i>	<code>MQRFH_STRUC_ID</code>	'RFHb'
<i>Version</i>	<code>MQRFH_VERSION_2</code>	2
<i>StrucLength</i>	<code>MQRFH_STRUC_LENGTH_FIXED_2</code>	36
<i>Encoding</i>	<code>MQENC_NATIVE</code>	Depends on environment
<i>CodedCharSetId</i>	<code>MQCCSI_INHERIT</code>	-2
<i>Format</i>	<code>MQFMT_NONE</code>	Blanks

Table 66. Initial values of fields in MQRFH2 for MQRFH2 (continued)

Field name	Name of constant	Value of constant
Flags	MQRFH_NONE	0
NameValueCCSID	None	1208
Notes: 1. The symbol b represents a single blank character. 2. In the C programming language, the macro variable MQRFH2_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure: <pre>MQRFH2 MyRFH2 = {MQRFH2_DEFAULT};</pre>		

C declaration

```
typedef struct tagMQRFH2 MQRFH2;
struct tagMQRFH2 {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQLONG    StrucLength;      /* Total length of MQRFH2 including all
                               NameValueLength and NameValueData
                               fields */
    MQLONG    Encoding;        /* Numeric encoding of data that follows
                               last NameValueData field */
    MQLONG    CodedCharSetId;   /* Character set identifier of data that
                               follows last NameValueData field */
    MQCHAR8   Format;          /* Format name of data that follows last
                               NameValueData field */
    MQLONG    Flags;           /* Flags */
    MQLONG    NameValueCCSID;   /* Character set identifier of
                               NameValueData */
};
```

COBOL declaration

```
** MQRFH2 structure
  10 MQRFH2.
** Structure identifier
  15 MQRFH2-STRUCID PIC X(4).
** Structure version number
  15 MQRFH2-VERSION PIC S9(9) BINARY.
** Total length of MQRFH2 including all NAMEVALUELENGTH and
** NAMEVALUEDATA fields
  15 MQRFH2-STRUCLength PIC S9(9) BINARY.
** Numeric encoding of data that follows last NAMEVALUEDATA field
  15 MQRFH2-ENCODING PIC S9(9) BINARY.
** Character set identifier of data that follows last NAMEVALUEDATA
** field
  15 MQRFH2-CODEDCHARSETID PIC S9(9) BINARY.
** Format name of data that follows last NAMEVALUEDATA field
  15 MQRFH2-FORMAT PIC X(8).
** Flags
  15 MQRFH2-FLAGS PIC S9(9) BINARY.
** Character set identifier of NAMEVALUEDATA
  15 MQRFH2-NAMEVALUECCSID PIC S9(9) BINARY.
```

PL/I declaration

```
dcl
  1 MQRFH2 based,
  3 StrucId      char(4),          /* Structure identifier */
  3 Version      fixed bin(31), /* Structure version number */
  3 StrucLength  fixed bin(31), /* Total length of MQRFH2 including
```

```

all NameValueLength and
NameValueData fields */
3 Encoding      fixed bin(31), /* Numeric encoding of data that
follows last NameValueData field */
3 CodedCharSetId fixed bin(31), /* Character set identifier of data
that follows last NameValueData
field */
3 Format        char(8),      /* Format name of data that follows
last NameValueData field */
3 Flags        fixed bin(31), /* Flags */
3 NameValueCCSID fixed bin(31); /* Character set identifier of
NameValueData */

```

System/390 assembler declaration

```

MQRFH          DSECT
MQRFH_STRUCID  DS   CL4  Structure identifier
MQRFH_VERSION  DS   F    Structure version number
MQRFH_STRUCLNGTH DS   F    Total length of MQRFH2 including all
*                NAMEVALUELENGTH and NAMEVALUEDATA fields
MQRFH_ENCODING DS   F    Numeric encoding of data that follows
*                last NAMEVALUEDATA field
MQRFH_CODEDCHARSETID DS   F    Character set identifier of data that
*                follows last NAMEVALUEDATA field
MQRFH_FORMAT   DS   CL8  Format name of data that follows last
*                NAMEVALUEDATA field
MQRFH_FLAGS    DS   F    Flags
MQRFH_NAMEVALUECCSID DS   F    Character set identifier of
*                NAMEVALUEDATA
*
MQRFH_LENGTH   EQU   *-MQRFH
                ORG   MQRFH
MQRFH_AREA     DS   CL(MQRFH_LENGTH)

```

Visual Basic declaration

```

Type MQRFH2
  StrucId      As String*4 'Structure identifier'
  Version      As Long     'Structure version number'
  StrucLength  As Long     'Total length of MQRFH2 including all'
                        'NameValueLength and NameValueData fields'
  Encoding     As Long     'Numeric encoding of data that follows'
                        'last NameValueData field'
  CodedCharSetId As Long   'Character set identifier of data that'
                        'follows last NameValueData field'
  Format       As String*8 'Format name of data that follows last'
                        'NameValueData field'
  Flags       As Long     'Flags'
  NameValueCCSID As Long   'Character set identifier of NameValueData'
End Type

```

MQRMH – Reference message header

The following table summarizes the fields in the structure.

Table 67. Fields in MQRMH

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>StrucLength</i>	Total length of MQRMH, including strings at end of fixed fields, but not the bulk data	StrucLength
<i>Encoding</i>	Numeric encoding of bulk data	Encoding

Table 67. Fields in MQRMH (continued)

Field	Description	Topic
<i>CodedCharSetId</i>	Character set identifier of bulk data	CodedCharSetId
<i>Format</i>	Format name of bulk data	Format
<i>Flags</i>	Reference message flags	Flags
<i>ObjectType</i>	Object type	ObjectType
<i>ObjectInstanceId</i>	Object instance identifier	ObjectInstanceId
<i>SrcEnvLength</i>	Length of source environment data	SrcEnvLength
<i>SrcEnvOffset</i>	Offset of source environment data	SrcEnvOffset
<i>SrcNameLength</i>	Length of source object name	SrcNameLength
<i>SrcNameOffset</i>	Offset of source object name	SrcNameOffset
<i>DestEnvLength</i>	Length of destination environment data	DestEnvLength
<i>DestEnvOffset</i>	Offset of destination environment data	DestEnvOffset
<i>DestNameLength</i>	Length of destination object name	DestNameLength
<i>DestNameOffset</i>	Offset of destination object name	DestNameOffset
<i>DataLogicalLength</i>	Length of bulk data	DataLogicalLength
<i>DataLogicalOffset</i>	Low offset of bulk data	DataLogicalOffset
<i>DataLogicalOffset2</i>	High offset of bulk data	DataLogicalOffset2

Overview for MQRMH

Availability: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Purpose: The MQRMH structure defines the format of a reference message header. This header is used with user-written message channel exits to send extremely large amounts of data (called *bulk data*) from one queue manager to another. The difference compared to normal messaging is that the bulk data is not stored on a queue; instead, only a *reference* to the bulk data is stored on the queue. This reduces the possibility of MQ resources being exhausted by a small number of extremely large messages.

Format name: MQFMT_REF_MSG_HEADER.

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

Set the character set and encoding of the MQRMH into the *CodedCharSetId* and *Encoding* fields in:

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

Usage: An application puts a message consisting of an MQRMH, but omitting the bulk data. When a message channel agent (MCA) reads the message from the transmission queue, 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 must exist. When a reference message is received, the exit must create the object from the bulk data that follows the MQRMH in the message, and then pass on the reference message without the bulk data. The reference message can later be retrieved by an application reading the reference message (without the bulk data) from a queue.

Normally, the MQRMH structure is all that is in the message. However, if the message is on a transmission queue, one or more additional headers 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: Do not send a reference message as a segmented message, because the message exit cannot process it correctly.

Data conversion: For data conversion purposes, converting the MQRMH structure includes conversion of the source environment data, source object name, destination environment data, and destination object name. Any other bytes within *StrucLength* bytes of the start of the structure are either discarded or have undefined values after data conversion. The bulk data is converted provided that all the following are true:

- The bulk data is present in the message when the data conversion is performed.
- The *Format* field in MQRMH has a value other than MQFMT_NONE.
- A user-written data-conversion exit exists with the format name specified.

Be aware, however, that usually the bulk data is *not* present in the message when the message is on a queue, and that as a result the bulk data is converted by the MQGMO_CONVERT option.

Fields for MQRMH

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

CodedCharSetId (MQLONG)

This specifies the character set identifier of the bulk data; it does not apply to character data in the MQRMH structure itself.

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

MQCCSI_INHERIT

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

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

Do not use MQCCSI_INHERIT if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

This value is supported in the following environments: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

The initial value of this field is MQCCSI_UNDEFINED.

DataLogicalLength (MQLONG)

The *DataLogicalLength* field specifies the length of the bulk data referenced by the MQRMH structure.

If the bulk data is actually present in the message, the data begins at an offset of *StrucLength* bytes from the start of the MQRMH structure. The length of the entire message minus *StrucLength* gives the length of the bulk data present.

If data is present in the message, *DataLogicalLength* specifies the amount of that data that is relevant. The normal case is for *DataLogicalLength* to have the same value as the length of data present in the message.

If the MQRMH structure represents the remaining data in the object (starting from the specified logical offset), you can use the value zero for *DataLogicalLength*, provided that the bulk data is not actually present in the message.

If no data is present, the end of MQRMH coincides with the end of the message.

The initial value of this field is 0.

DataLogicalOffset (MQLONG)

This field specifies the low offset of the bulk data from the start of the object of which the bulk data forms part. The offset of the bulk data from the start of the object is called the *logical offset*. This is *not* the physical offset of the bulk data from the start of the MQRMH structure; that offset is given by *StrucLength*.

To allow large objects to be sent using reference messages, the logical offset is divided into two fields, and the actual logical offset is given by the sum of these two fields:

- *DataLogicalOffset* represents the remainder obtained when the logical offset is divided by 1 000 000 000. It is thus a value in the range 0 through 999 999 999.
- *DataLogicalOffset2* represents the result obtained when the logical offset is divided by 1 000 000 000. It is thus the number of complete multiples of 1 000 000 000 that exist in the logical offset. The number of multiples is in the range 0 through 999 999 999.

The initial value of this field is 0.

DataLogicalOffset2 (MQLONG)

This field specifies the high offset of the bulk data from the start of the object of which the bulk data forms part. It is a value in the range 0 through 999 999 999. See *DataLogicalOffset* for details.

The initial value of this field is 0.

DestEnvLength (MQLONG)

This is the length of the destination environment data. If this field is zero, there is no destination environment data, and *DestEnvOffset* is ignored.

DestEnvOffset (MQLONG)

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 Windows the destination environment data might be the directory path of the object where the bulk data is to be stored. However, if the creator does not know the destination environment data, it is the responsibility of the user-supplied message exit to determine any environment information needed.

The length of the destination environment data is given by *DestEnvLength*; if this length is zero, there is no destination environment data, and *DestEnvOffset* is ignored. If present, the destination environment data must reside completely within *StrucLength* bytes from the start of the structure.

Applications must not assume that the destination environment data is contiguous with any of the data addressed by the *SrcEnvOffset*, *SrcNameOffset*, and *DestNameOffset* fields.

The initial value of this field is 0.

DestNameLength (MQLONG)

The length of the destination object name. If this field is zero, there is no destination object name, and *DestNameOffset* is ignored.

DestNameOffset (MQLONG)

This field specifies the offset of the destination object name from the start of the MQRMH structure. The destination object name can be specified by the creator of the reference message, if that data is known to the creator. However, if the creator does not know the destination object name, it is the responsibility of the user-supplied message exit to identify the object to be created or modified.

The length of the destination object name is given by *DestNameLength*; if this length is zero, there is no destination object name, and *DestNameOffset* is ignored. If present, the destination object name must reside completely within *StrucLength* bytes from the start of the structure.

Applications must not assume that the destination object name is contiguous with any of the data addressed by the *SrcEnvOffset*, *SrcNameOffset*, and *DestEnvOffset* fields.

The initial value of this field is 0.

Encoding (MQLONG)

This specifies the numeric encoding of the bulk data; it does not apply to numeric data in the MQRMH structure itself.

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

The initial value of this field is MQENC_NATIVE.

Flags (MQLONG)

These are reference message flags. The following flags are defined:

MQRMHF_LAST

This flag indicates that the reference message represents or contains the last part of the referenced object.

MQRMHF_NOT_LAST

Reference message does not contain or represent last part of object. MQRMHF_NOT_LAST aids program documentation. It is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

The initial value of this field is MQRMHF_NOT_LAST.

Format (MQCHAR8)

This specifies the format name of the bulk data.

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

The initial value of this field is MQFMT_NONE.

ObjectInstanceId (MQBYTE24)

Use this field to identify a specific instance of an object. If it is not needed, set it to the following value:

MQOII_NONE

No object instance identifier specified. The value is binary zero for the length of the field.

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

The length of this field is given by MQ_OBJECT_INSTANCE_ID_LENGTH. The initial value of this field is MQOII_NONE.

ObjectType (MQCHAR8)

This is a name that the message exit can use to recognize types of reference message that it supports. The name must conform to the same rules as the *Format* field described above.

The initial value of this field is 8 blanks.

SrcEnvLength (MQLONG)

The length of the source environment data. If this field is zero, there is no source environment data, and *SrcEnvOffset* is ignored.

The initial value of this field is 0.

SrcEnvOffset (MQLONG)

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 Windows 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, the user-supplied message exit must determine any environment information needed.

The length of the source environment data is given by *SrcEnvLength*; if this length is zero, there is no source environment data, and *SrcEnvOffset* is ignored. If present, the source environment data must reside completely within *StrucLength* bytes from the start of the structure.

Applications must not assume that the environment data starts immediately after the last fixed field in the structure or that it is contiguous with any of the data addressed by the *SrcNameOffset*, *DestEnvOffset*, and *DestNameOffset* fields.

The initial value of this field is 0.

SrcNameLength (MQLONG)

The length of the source object name. If this field is zero, there is no source object name, and *SrcNameOffset* is ignored.

The initial value of this field is 0.

SrcNameOffset (MQLONG)

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, the user-supplied message exit must identify the object to be accessed.

The length of the source object name is given by *SrcNameLength*; if this length is zero, there is no source object name, and *SrcNameOffset* is ignored. If present, the source object name must reside completely within *StrucLength* bytes from the start of the structure.

Applications must not assume that the source object name is contiguous with any of the data addressed by the *SrcEnvOffset*, *DestEnvOffset*, and *DestNameOffset* fields.

The initial value of this field is 0.

StrucId (MQCHAR4)

This is the structure identifier; the value must be:

MQRMH_STRUC_ID

Identifier for reference message header structure.

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

The initial value of this field is MQRMH_STRUC_ID.

StrucLength (MQLONG)

The total length of MQRMH, including strings at the end of fixed fields, but not the bulk data.

The initial value of this field is zero.

Version (MQLONG)

The structure version number. The value must be:

MQRMH_VERSION_1

Version-1 reference message header structure.

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

MQRMH_CURRENT_VERSION

Current version of reference message header structure.

The initial value of this field is MQRMH_VERSION_1.

Initial values and language declarations for MQRMH

Table 68. Initial values of fields in MQRMH for MQRMH

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

Table 68. Initial values of fields in MQRMH for MQRMH (continued)

Field name	Name of constant	Value of constant
Notes:		
1. The symbol b represents a single blank character.		
2. In the C programming language, the macro variable MQRMH_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:		
MQRMH MyRMH = {MQRMH_DEFAULT};		

C declaration

```
typedef struct tagMQRMH MQRMH;
struct tagMQRMH {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQLONG    StrucLength;      /* Total length of MQRMH, including
                               strings at end of fixed fields, but
                               not the bulk data */

    MQLONG    Encoding;         /* Numeric encoding of bulk data */
    MQLONG    CodedCharSetId;   /* Character set identifier of bulk
                               data */

    MQCHAR8   Format;           /* Format name of bulk data */
    MQLONG    Flags;            /* Reference message flags */
    MQCHAR8   ObjectType;       /* Object type */
    MQBYTE24  ObjectInstanceId; /* Object instance identifier */
    MQLONG    SrcEnvLength;     /* Length of source environment data */
    MQLONG    SrcEnvOffset;     /* Offset of source environment data */
    MQLONG    SrcNameLength;    /* Length of source object name */
    MQLONG    SrcNameOffset;    /* Offset of source object name */
    MQLONG    DestEnvLength;    /* Length of destination environment
                               data */
    MQLONG    DestEnvOffset;    /* Offset of destination environment
                               data */

    MQLONG    DestNameLength;   /* Length of destination object name */
    MQLONG    DestNameOffset;   /* Offset of destination object name */
    MQLONG    DataLogicalLength; /* Length of bulk data */
    MQLONG    DataLogicalOffset; /* Low offset of bulk data */
    MQLONG    DataLogicalOffset2; /* High offset of bulk data */
};
```

COBOL declaration

```
** MQRMH structure
   10 MQRMH.
**   Structure identifier
   15 MQRMH-STRUCID          PIC X(4).
**   Structure version number
   15 MQRMH-VERSION         PIC S9(9) BINARY.
**   Total length of MQRMH, including strings at end of fixed fields,
**   but not the bulk data
   15 MQRMH-STRUCLength    PIC S9(9) BINARY.
**   Numeric encoding of bulk data
   15 MQRMH-ENCODING        PIC S9(9) BINARY.
**   Character set identifier of bulk data
   15 MQRMH-CODEDCHARSETID PIC S9(9) BINARY.
**   Format name of bulk data
   15 MQRMH-FORMAT          PIC X(8).
**   Reference message flags
   15 MQRMH-FLAGS           PIC S9(9) BINARY.
**   Object type
   15 MQRMH-OBJECTTYPE      PIC X(8).
**   Object instance identifier
```

```

15 MQRMH-OBJECTINSTANCEID PIC X(24).
** Length of source environment data
15 MQRMH-SRCENVLENGTH PIC S9(9) BINARY.
** Offset of source environment data
15 MQRMH-SRCENVOFFSET PIC S9(9) BINARY.
** Length of source object name
15 MQRMH-SRCNAMELENGTH PIC S9(9) BINARY.
** Offset of source object name
15 MQRMH-SRCNAMEOFFSET PIC S9(9) BINARY.
** Length of destination environment data
15 MQRMH-DESTENVLENGTH PIC S9(9) BINARY.
** Offset of destination environment data
15 MQRMH-DESTENVOFFSET PIC S9(9) BINARY.
** Length of destination object name
15 MQRMH-DESTNAMELENGTH PIC S9(9) BINARY.
** Offset of destination object name
15 MQRMH-DESTNAMEOFFSET PIC S9(9) BINARY.
** Length of bulk data
15 MQRMH-DATALOGICALENGTH PIC S9(9) BINARY.
** Low offset of bulk data
15 MQRMH-DATALOGICALOFFSET PIC S9(9) BINARY.
** High offset of bulk data
15 MQRMH-DATALOGICALOFFSET2 PIC S9(9) BINARY.

```

PL/I declaration

```

dcl
1 MQRMH based,
3 StrucId char(4), /* Structure identifier */
3 Version fixed bin(31), /* Structure version number */
3 StrucLength fixed bin(31), /* Total length of MQRMH,
including strings at end of
fixed fields, but not the bulk
data */
3 Encoding fixed bin(31), /* Numeric encoding of bulk
data */
3 CodedCharSetId fixed bin(31), /* Character set identifier of
bulk data */
3 Format char(8), /* Format name of bulk data */
3 Flags fixed bin(31), /* Reference message flags */
3 ObjectType char(8), /* Object type */
3 ObjectInstanceId char(24), /* Object instance identifier */
3 SrcEnvLength fixed bin(31), /* Length of source environment
data */
3 SrcEnvOffset fixed bin(31), /* Offset of source environment
data */
3 SrcNameLength fixed bin(31), /* Length of source object name */
3 SrcNameOffset fixed bin(31), /* Offset of source object name */
3 DestEnvLength fixed bin(31), /* Length of destination
environment data */
3 DestEnvOffset fixed bin(31), /* Offset of destination
environment data */
3 DestNameLength fixed bin(31), /* Length of destination object
name */
3 DestNameOffset fixed bin(31), /* Offset of destination object
name */
3 DataLogicalLength fixed bin(31), /* Length of bulk data */
3 DataLogicalOffset fixed bin(31), /* Low offset of bulk data */
3 DataLogicalOffset2 fixed bin(31); /* High offset of bulk data */

```

System/390 assembler declaration

```

MQRMH DSECT
MQRMH_STRUCID DS CL4 Structure identifier
MQRMH_VERSION DS F Structure version number
MQRMH_STRUCLNGTH DS F Total length of MQRMH, including
* strings at end of fixed fields, but
* not the bulk data

```



```

MQRMH_ENCODING          DS  F    Numeric encoding of bulk data
MQRMH_CODEDCHARSETID   DS  F    Character set identifier of bulk
*                          data
MQRMH_FORMAT            DS  CL8   Format name of bulk data
MQRMH_FLAGS             DS  F    Reference message flags
MQRMH_OBJECTTYPE       DS  CL8   Object type
MQRMH_OBJECTINSTANCEID DS  XL24  Object instance identifier
MQRMH_SRCENVLENGTH     DS  F    Length of source environment data
MQRMH_SRCENVOFFSET     DS  F    Offset of source environment data
MQRMH_SRCNAMELENGTH    DS  F    Length of source object name
MQRMH_SRCNAMEOFFSET    DS  F    Offset of source object name
MQRMH_DESTENVLENGTH    DS  F    Length of destination environment
*                          data
MQRMH_DESTENVOFFSET    DS  F    Offset of destination environment
*                          data
MQRMH_DESTNAMELENGTH   DS  F    Length of destination object name
MQRMH_DESTNAMEOFFSET   DS  F    Offset of destination object name
MQRMH_DATALOGICALENGTH DS  F    Length of bulk data
MQRMH_DATALOGICALOFFSET DS  F    Low offset of bulk data
MQRMH_DATALOGICALOFFSET2 DS  F    High offset of bulk data
*
MQRMH_LENGTH           EQU  *-MQRMH
                        ORG  MQRMH
MQRMH_AREA             DS  CL(MQRMH_LENGTH)

```

Visual Basic declaration

```

Type MQRMH
  StrucId          As String*4 'Structure identifier'
  Version          As Long     'Structure version number'
  StrucLength      As Long     'Total length of MQRMH, including
                              'strings at end of fixed fields, but'
                              'not the bulk data'

  Encoding         As Long     'Numeric encoding of bulk data'
  CodedCharSetId  As Long     'Character set identifier of bulk data'
  Format           As String*8 'Format name of bulk data'
  Flags           As Long     'Reference message flags'
  ObjectType       As String*8 'Object type'
  ObjectInstanceId As MQBYTE24 'Object instance identifier'
  SrcEnvLength     As Long     'Length of source environment data'
  SrcEnvOffset     As Long     'Offset of source environment data'
  SrcNameLength   As Long     'Length of source object name'
  SrcNameOffset   As Long     'Offset of source object name'
  DestEnvLength   As Long     'Length of destination environment'
                              'data'
  DestEnvOffset   As Long     'Offset of destination environment'
                              'data'

  DestNameLength  As Long     'Length of destination object name'
  DestNameOffset  As Long     'Offset of destination object name'
  DataLogicalLength As Long     'Length of bulk data'
  DataLogicalOffset As Long     'Low offset of bulk data'
  DataLogicalOffset2 As Long     'High offset of bulk data'
End Type

```

MQRR – Response record

The following table summarizes the fields in the structure.

Table 69. Fields in MQRR

Field	Description	Topic
<i>CompCode</i>	Completion code for queue	CompCode
<i>Reason</i>	Reason code for queue	Reason

Overview for MQRR

Availability: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Purpose: Use the MQRR structure to receive the completion code and reason code resulting from the open or put operation for a single destination queue, when the destination is a distribution list. MQRR is an output structure for the MQOPEN, MQPUT, and MQPUT1 calls.

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

Usage: By providing an array of these structures on the MQOPEN and MQPUT calls, or on the MQPUT1 call, you can determine the completion codes and reason codes for all the queues in a distribution list when the outcome of the call is mixed, that is, when the call succeeds for some queues in the list but fails for others. Reason code MQRC_MULTIPLE_REASONS from the call indicates that the response records (if provided by the application) have been set by the queue manager.

Fields for MQRR

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

CompCode (MQLONG)

This is the completion code resulting from the open or put operation for the queue whose name was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call.

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

Reason (MQLONG)

This is the reason code resulting from the open or put operation for the queue whose name was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call.

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

Initial values and language declarations for MQRR

Table 70. Initial values of fields in MQRR for MQRR

Field name	Name of constant	Value of constant
<i>CompCode</i>	MQCC_OK	0
<i>Reason</i>	MQRC_NONE	0

Table 70. Initial values of fields in MQRR for MQRR (continued)

Field name	Name of constant	Value of constant
Notes:		
1. In the C programming language, the macro variable MQRR_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure: MQRR MyRR = {MQRR_DEFAULT};		

C declaration

```
typedef struct tagMQRR MQRR;
struct tagMQRR {
    MQLONG CompCode; /* Completion code for queue */
    MQLONG Reason; /* Reason code for queue */
};
```

COBOL declaration

```
** MQRR structure
 10 MQRR.
** Completion code for queue
 15 MQRR-COMPCODE PIC S9(9) BINARY.
** Reason code for queue
 15 MQRR-REASON PIC S9(9) BINARY.
```

PL/I declaration

```
dcl
 1 MQRR based,
 3 CompCode fixed bin(31), /* Completion code for queue */
 3 Reason fixed bin(31); /* Reason code for queue */
```

Visual Basic declaration

```
Type MQRR
    CompCode As Long 'Completion code for queue'
    Reason As Long 'Reason code for queue'
End Type
```

MQSCO – SSL configuration options

The following table summarizes the fields in the structure.

Table 71. Fields in MQSCO

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>KeyRepository</i>	Location of key repository	KeyRepository
<i>CryptoHardware</i>	Details of cryptographic hardware	CryptoHardware
<i>AuthInfoRecCount</i>	Number of MQAIR records present	AuthInfoRecCount
<i>AuthInfoRecOffset</i>	Offset of first MQAIR record from start of MQSCO	AuthInfoRecOffset
<i>AuthInfoRecPtr</i>	Address of first MQAIR record	AuthInfoRecPtr
Note: The remaining fields are ignored if <i>Version</i> is less than MQSCO_VERSION_2.		
<i>KeyResetCount</i>	SSL secret key reset count	KeyResetCount

Table 71. Fields in MQSCO (continued)

Field	Description	Topic
<i>Fips Required</i>	Use FIPS-certified cryptographic algorithms in WebSphere MQ	"FipsRequired (MQLONG)" on page 330

Overview for MQSCO

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

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

If the channel protocol for the client channel is not TCP/IP, the MQSCO structure is ignored.

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

Fields for MQSCO

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

AuthInfoRecCount (MQLONG)

This is the number of authentication information (MQAIR) records addressed by the *AuthInfoRecPtr* or *AuthInfoRecOffset* fields. For more information, see "MQAIR – Authentication information record" on page 29. The value must be zero or greater. If the value is not valid, the call fails with reason code MQRC_AUTH_INFO_REC_COUNT_ERROR.

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

AuthInfoRecOffset (MQLONG)

This is the offset in bytes of the first authentication information record from the start of the MQSCO structure. The offset can be positive or negative. The field is ignored if *AuthInfoRecCount* is zero.

You can use either *AuthInfoRecOffset* or *AuthInfoRecPtr* to specify the MQAIR records, but not both; see the description of the *AuthInfoRecPtr* field for details.

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

AuthInfoRecPtr (PMQAIR)

This is the address of the first authentication information record. The field is ignored if *AuthInfoRecCount* is zero.

You can provide the array of MQAIR records in one of two ways:

- By using the pointer field *AuthInfoRecPtr*
In this case, the application can declare an array of MQAIR records that is separate from the MQSCO structure, and set *AuthInfoRecPtr* to the address of the array.
Using *AuthInfoRecPtr* is recommended for programming languages that support the pointer data type in a fashion that is portable to different environments (for example, the C programming language).
- By using the offset field *AuthInfoRecOffset*
In this case, the application must declare a compound structure containing an MQSCO followed by the array of MQAIR records, and set *AuthInfoRecOffset* to the offset of the first record in the array from the start of the MQSCO structure. Ensure that this value is correct, and has a value that can be accommodated within an MQLONG (the most restrictive programming language is COBOL, for which the valid range is -999 999 999 through +999 999 999).
Using *AuthInfoRecOffset* is recommended for programming languages that do not support the pointer data type, or that implement the pointer data type in a fashion that is not portable to different environments (for example, the COBOL programming language).

Whatever technique you choose, only one of *AuthInfoRecPtr* and *AuthInfoRecOffset* can be used; the call fails with reason code MQRC_AUTH_INFO_REC_ERROR if both are nonzero.

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

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

CryptoHardware (MQCHAR256)

This field gives configuration details for cryptographic hardware connected to the client system.

Set the field to a string of the following format, or leave it blank or null:

```
GSK_PKCS11=<the PKCS #11 driver path and filename>;<the PKCS #11
token label>;<the PKCS #11 token password>;<symmetric cipher setting>;
```

To use cryptographic hardware which conforms to the PKCS11 interface, for example, the IBM 4960 or IBM 4764, the PKCS11 driver path, PKCS11 token label, and PKCS11 token password strings must be specified, each terminated by a semi-colon.

The PKCS #11 driver path is an absolute path to the shared library providing support for the PKCS #11 card. The PKCS #11 driver filename is the name of the shared library. An example of the value required for the PKCS #11 path and filename is:

```
/usr/lib/pkcs11/PKCS11_API.so
```

The PKCS #11 token label must be entirely in lowercase. If you have configured your hardware with a mixed case or uppercase token label, re-configure it with this lowercase label.

If no cryptographic hardware configuration is required, set the field to blank or null.

If the value is shorter than the length of the field, terminate the value with a null character, or pad it with blanks to the length of the field. If the value is not valid, or leads to a failure when used to configure the cryptographic hardware, the call fails with reason code MQRC_CRYPTO_HARDWARE_ERROR.

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

FipsRequired (MQLONG)

WebSphere MQ can be configured with cryptographic hardware so that the cryptography modules used are those provided by the hardware product; these can either be FIPS-certified, or not, to a particular level depending on the cryptographic hardware product in use. Use this field to specify that only FIPS-certified algorithms are used if the cryptography is provided in WebSphere MQ-provided software.

When WebSphere MQ is installed an implementation of SSL cryptography is also installed which provides some FIPS-certified modules.

The values can be:

MQSSL_FIPS_NO

This is the default value. When set to this value:

- Any CipherSpec supported on a particular platform can be used.
- If run without use of cryptographic hardware, the following CipherSpecs run using FIPS 140-2 certified cryptography on the WebSphere MQ platforms:
 - TLS_RSA_WITH_3DES_EDE_CBC_SHA
 - FIPS_WITH_3DES_EDE_CBC_SHA
 - TLS_RSA_WITH_AES_128_CBC_SHA
 - TLS_RSA_WITH_AES_256_CBC_SHA

MQSSL_FIPS_YES

When set to this value, unless you are using cryptographic hardware to perform the cryptography, you can be sure that

- Only FIPS-certified cryptographic algorithms can be used in the CipherSpec applying to this client connection.
- Inbound and outbound SSL channel connections only succeed if one of the following Cipher Specs are used:
 - TLS_RSA_WITH_3DES_EDE_CBC_SHA
 - FIPS_WITH_3DES_EDE_CBC_SHA
 - TLS_RSA_WITH_AES_128_CBC_SHA
 - TLS_RSA_WITH_AES_256_CBC_SHA

KeyRepository (MQCHAR256)

This field is relevant only for WebSphere MQ clients running on UNIX systems and Windows systems. It specifies the location of the key database file in which keys and certificates are stored. The key database file must have a file name of the form *zzz.kdb*, where *zzz* is user-selectable. The *KeyRepository* field contains the path to this file, along with the file name stem (all characters in the file name up to but not including the final *.kdb*). The *.kdb* file suffix is added automatically.

Each key database file has an associated *password stash file*. This holds encrypted passwords that are used to allow programmatic access to the key database. The password stash file must reside in the same directory and have the same file stem as the key database, and must end with the suffix `.sth`.

For example, if the *KeyRepository* field has the value `/xxx/yyy/key`, the key database file must be `/xxx/yyy/key.kdb`, and the password stash file must be `/xxx/yyy/key.sth`, where `xxx` and `yyy` represent directory names.

If the value is shorter than the length of the field, terminate the value with a null character, or pad it with blanks to the length of the field. The value is not checked; if there is an error in accessing the key repository, the call fails with reason code `MQRC_KEY_REPOSITORY_ERROR`.

To run an SSL connection from a WebSphere MQ client, set *KeyRepository* to a valid key database file name.

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

KeyResetCount (MQLONG)

This represents the total number of unencrypted bytes sent and received within an SSL conversation before the secret key is renegotiated. The number of bytes includes control information sent by the MCA.

If you specify an SSL/TLS secret key reset count between 1 byte and 32Kb, SSL/TLS channels will use a secret key reset count of 32Kb. This is to avoid the overhead of excessive key resets which would occur for small SSL/TLS secret key reset values.

This is an input field. The value is a number between 0 and 999 999 999, with a default value of 0. Use a value of 0 to indicate that secret keys are never renegotiated.

StrucId (MQCHAR4)

This is the structure identifier; the value must be:

MQSCO_STRUC_ID

Identifier for SSL configuration options structure.

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

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

Version (MQLONG)

This is the structure version number; the value must be:

MQSCO_VERSION_1

Version-1 SSL configuration options structure.

MQSCO_VERSION_2

Version-2 SSL configuration options structure.

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

MQSCO_CURRENT_VERSION

Current version of SSL configuration options structure.

This is always an input field. The initial value of this field is MQSCO_VERSION_2

Initial values and language declarations for MQSCO

Table 72. Initial values of fields in MQSCO

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQSCO_STRUC_ID	'SC0b'
<i>Version</i>	MQSCO_CURRENT_VERSION	1
<i>KeyRepository</i>	None	Null string or blanks
<i>CryptoHardware</i>	None	Null string or blanks
<i>AuthInfoRecCount</i>	None	0
<i>AuthInfoRecOffset</i>	None	0
<i>AuthInfoRecPtr</i>	None	Null pointer or null bytes

Notes:

1. The symbol b represents a single blank character.
2. In the C programming language, the macro variable MQSCO_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

```
MQSCO MySCO = {MQSCO_DEFAULT};
```

C declaration

```
typedef struct tagMQSCO MQSCO;
struct tagMQSCO {
    MQCHAR4    StrucId;           /* Structure identifier */
    MQLONG     Version;          /* Structure version number */
    MQCHAR256  KeyRepository;    /* Location of SSL key repository */
    MQCHAR256  CryptoHardware;   /* Cryptographic hardware configuration
                                string */
    MQLONG     AuthInfoRecCount; /* Number of MQAIR records present */
    MQLONG     AuthInfoRecOffset; /* Offset of first MQAIR record from
                                start of MQSCO structure */
    PMQAIR     AuthInfoRecPtr;   /* Address of first MQAIR record */
};
```

COBOL declaration

```
** MQSCO structure
10 MQSCO.
** Structure identifier
15 MQSCO-STRUCID PIC X(4).
** Structure version number
15 MQSCO-VERSION PIC S9(9) BINARY.
** Location of SSL key repository
15 MQSCO-KEYREPOSITORY PIC X(256).
** Cryptographic hardware configuration string
15 MQSCO-CRYPTOHardware PIC X(256).
** Number of MQAIR records present
15 MQSCO-AUTHINFORECCOUNT PIC S9(9) BINARY.
```

```

**      Offset of first MQAIR record from start of MQSCO structure
      15 MQSCO-AUTHINFORECOFFSET PIC S9(9) BINARY.
**      Address of first MQAIR record
      15 MQSCO-AUTHINFORECPtr    POINTER.

```

PL/I declaration

```

dcl
1 MQSCO based,
  3 StrucId          char(4),          /* Structure identifier */
  3 Version          fixed bin(31),    /* Structure version number */
  3 KeyRepository    char(256),        /* Location of SSL key
                                        repository */
  3 CryptoHardware   char(256),        /* Cryptographic hardware
                                        configuration string */
  3 AuthInfoRecCount fixed bin(31),    /* Number of MQAIR records
                                        present */
  3 AuthInfoRecOffset fixed bin(31),   /* Offset of first MQAIR record
                                        from start of MQSCO structure */
  3 AuthInfoRecPtr   pointer;          /* Address of first MQAIR record */

```

Visual Basic declaration

```

Type MQSCO
  StrucId          As String*4        'Structure identifier'
  Version          As Long            'Structure version number'
  KeyRepository    As String*256     'Location of SSL key repository'
  CryptoHardware   As String*256     'Cryptographic hardware configuration'
                                        'string'
  AuthInfoRecCount As Long            'Number of MQAIR records present'
  AuthInfoRecOffset As Long          'Offset of first MQAIR record from'
                                        'start of MQSCO structure'
  AuthInfoRecPtr   As MQPTR          'Address of first MQAIR record'
End Type

```

MQSD - Subscription descriptor

The following table summarizes the fields in the structure.

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>Options</i>	Options	Options
<i>ObjectName</i>	Object name	ObjectName
<i>AlternateUserId</i>	Alternate User Id	AlternateUserId
<i>AlternateSecurityId</i>	Alternate Security Id	AlternateSecurityId
<i>SubExpiry</i>	Subscription Expiry	SubExpiry
<i>ObjectString</i>	Object String	ObjectString
<i>SubName</i>	Subscription Name	SubName
<i>SubUserData</i>	Subscription user data	SubUserData
<i>SubCorrelId</i>	Subscription Correlation Id	SubCorrelId
<i>PubPriority</i>	Publication priority	PubPriority
<i>PubAccountingToken</i>	Publication Accounting Token	PubAccountingToken
<i>PubAppIdentityData</i>	Publication application identity data	PubAppIdentityData
<i>SelectionString</i>	String providing selection criteria	SelectionString
<i>SubLevel</i>	Subscription Level	SubLevel

Field	Description	Topic
<i>ResObjectString</i>	Long object name	ResObjectString

Overview for MQSD

Availability: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, z/OS, plus WebSphere MQ clients connected to these systems.

Purpose: The MQSD structure is used to specify details about the subscription being made.

The structure is an input/output parameter on the MQSUB call for more information see “Usage notes for MQSUB” on page 568

Managed subscriptions: If an application has no specific need to use a particular queue as the destination for those publications that match its subscription, it can make use of the managed subscription feature. If an application elects to use a managed subscription, the queue manager informs the subscriber about the destination where published messages will be sent, by providing an object handle as an output from the MQSUB call. For more information, see “Hobj (MQHOBJ) - Input/output” on page 565.

When the subscription is removed, the queue manager also undertakes to clean up messages that have not been retrieved from the managed destination, in the following situations:

- When the subscription is removed - by use of MQCLOSE with MQCO_REMOVE_SUB - and the managed Hobj is closed.
- By implicit means when the connection is lost to an application using a non-durable subscription (MQSO_NON_DURABLE)
- By expiration when a subscription is removed because it has expired and the managed Hobj is closed.

You must use managed subscriptions with non-durable subscriptions, so that this clean up can occur, and so that messages for closed non-durable subscriptions do not take up space in your queue manager. Durable subscriptions can also use managed destinations.

Version: The current version of MQSD is MQSD_VERSION_1.

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

Fields for MQSD

The MQSD structure contains the following fields; the fields are described in alphabetical order:

AlternateSecurityId (MQBYTE40)

This is a security identifier that is passed with the AlternateUserId to the authorization service to allow appropriate authorization checks to be performed.

AlternateSecurityId is used only if MQSO_ALTERNATE_USER_AUTHORITY is specified, and the AlternateUserId field is not entirely blank up to the first null character or the end of the field.

On return from an MQSUB call using MQSO_RESUME, this field is unchanged.

See the description of “AlternateSecurityId (MQBYTE40)” on page 251 in the MQOD data type for more information.

AlternateUserId

If you specify MQSO_ALTERNATE_USER_AUTHORITY, this field contains an alternate user identifier that is used to check the authorization for the subscription and for output to the destination queue (specified in the *Hobj* parameter of the MQSUB call), in place of the user identifier that the application is currently running under.

If successful, the user identifier specified in this field is recorded as the subscription owning user identifier in place of the user identifier that the application is currently running under.

If MQSO_ALTERNATE_USER_AUTHORITY is specified and this field is entirely blank up to the first null character or the end of the field, the subscription can succeed only if no user authorization is needed to subscribe to this topic with the options specified or the destination queue for output.

If MQSO_ALTERNATE_USER_AUTHORITY is not specified, this field is ignored.

The following differences exist in the environments indicated:

- On z/OS, only the first 8 characters of AlternateUserId are used to check the authorization for the subscription. However, the current user identifier must be authorized to specify this particular alternate user identifier; all 12 characters of the alternate user identifier are used for this check. The user identifier must contain only characters allowed by the external security manager.

On return from an MQSUB call using MQSO_RESUME, this field is unchanged.

This is an input field. The length of this field is given by MQ_USER_ID_LENGTH. The initial value of this field is the null string in C, and 12 blank characters in other programming languages.

ObjectName (MQCHAR48)

This is the name of the topic object as defined on the local queue manager.

The name can contain the following characters:

- Uppercase alphabetic characters (A through Z)
- Lowercase alphabetic characters (a through z)
- Numeric digits (0 through 9)
- Period (.), forward slash (/), underscore (_), percent (%)

The name must not contain leading or embedded blanks, but can contain trailing blanks. Use a null character to indicate the end of significant data in the name; the null and any characters following it are treated as blanks. The following restrictions apply in the environments indicated:

- On systems that use EBCDIC Katakana, lowercase characters cannot be used.
- On z/OS:
 - Avoid names that begin or end with an underscore; they cannot be processed by the operations and control panels.
 - The percent character has a special meaning to RACF. If RACF is used as the external security manager, names must not contain the percent. If they do, those names are not included in any security checks when RACF generic profiles are used.
- On i5/OS, names containing lowercase characters, forward slash, or percent, must be enclosed in quotation marks when specified on commands. These quotation marks must not be specified for names that occur as fields in structures or as parameters on calls.

The *ObjectName* is used to form the full topic name.

The full topic name can be built from two different fields: *ObjectName* and *ObjectString*. For details of how these two fields are used, see “Using topic strings” on page 351.

If the object identified by the *ObjectName* field cannot be found, the call fails with reason code MQRC_UNKNOWN_OBJECT_NAME even if there is a string specified in *ObjectString*.

| On return from an MQSUB call using the MQSO_RESUME option this field is
| unchanged.

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

| If altering an existing subscription using the MQSO_ALTER option, the name of
| the topic object subscribed to cannot be changed. This field and the *ObjectString*
| field can be omitted. If they are provided, they must resolve to the same full topic
| name. If they do not, the call fails with MQRC_TOPIC_NOT_ALTERABLE.

ObjectString (MQCHARV)

This is the long object name to be used.

The *ObjectString* is used to form the Full topic name.

The full topic name can be built from two different fields: *ObjectName* and *ObjectString*. For details of how these two fields are used, see “Using topic strings” on page 351.

| The maximum length of *ObjectString* is 10240.

| If *ObjectString* is not specified correctly, according to the description of how to
| use the MQCHARV structure, or if it exceeds the maximum length, the call fails
| with reason code MQRC_OBJECT_STRING_ERROR.

This is an input field. The initial values of the fields in this structure are the same as those in the MQCHARV structure.

If there are wildcards in the *ObjectString* the interpretation of those wildcards can be controlled using the Wildcard options specified in the Options field of the MQSD.

| On return from an MQSUB call using the MQSO_RESUME option this field is
 | unchanged. The full topic name used is returned in the *ResObjectString* field if a
 | buffer is provided.

| If altering an existing subscription using the MQSO_ALTER option, the long name
 | of the topic object subscribed to cannot be changed. This field and the *ObjectName*
 | field can be omitted. If they are provided they must resolve to the same full topic
 | name or the call fails with MQRC_TOPIC_NOT_ALTERABLE.

Options (MQLONG)

You must specify at least one of the following options: These options control the action of the MQSUB call.

- MQSO_ALTER
- MQSO_RESUME
- MQSO_CREATE

The values you specify for the options can be used in the following ways:

- The values can be added together. Do not add the same constant more than once
- The values can be combined using the bitwise OR operation, if the programming language supports bitwise operations.

Combinations that are not valid are noted in this topic; any other combinations are valid.

Access or creation options: Access and creation options control whether a subscription is created, or whether an existing subscription is returned or altered. You must specify at least one of these options. The table displays valid combinations of access and creation options.

Combination of options	Notes
MQSO_CREATE	Creates a subscription if one doesn't exist. This combination fails if the subscription already exists.
MQSO_RESUME	Resumes an existing subscription. This combination fails if no subscription exists.
MQSO_CREATE + MQSO_RESUME	Creates a subscription if one doesn't exist and resumes a matching one, if it does exist. This combination is useful when it is used in an application that will be run a number of times.
MQSO_ALTER (see note)	Resumes an existing subscription, altering any fields to match that specified in the MQSD. This combination fails if no subscription exists.
MQSO_CREATE + MQSO_ALTER (see note)	Creates a subscription if one doesn't exist and resumes a matching one, if it does exist, altering any fields to match that specified in the MQSD. This combination is useful combination when used in an application that wants to ensure its subscription is in a certain state before proceeding.

Combination of options	Notes
<p>Note:</p> <p>Options specifying MQSO_ALTER can also specify MQSO_RESUME, but this combination has no additional effect to specifying MQSO_ALTER alone. In other words, MQSO_ALTER implies MQSO_RESUME, because calling MQSUB to alter a subscription implies that the subscription will also be resumed. The opposite is not true, however: resuming a subscription does not imply it is to be altered.</p>	

MQSO_CREATE

Create a new subscription for the topic specified. If a subscription using the same *SubName* already exists, the call fails with MQRC_SUB_ALREADY_EXISTS. This failure can be avoided by combining the MQSO_CREATE option with MQSO_RESUME. The *SubName* is not always necessary. For more details see the description of that field.

Combining MQSO_CREATE with MQSO_RESUME returns a handle to a pre-existing subscription for the specified *SubName* if one is found; if there is no existing subscription, a new one is created using all the fields provided in the MQSD.

MQSO_CREATE can also be combined with MQSO_ALTER to similar effect.

MQSO_RESUME

Return a handle to a pre-existing subscription which matches that specified by *SubName*. No changes are made to the matching subscription's attributes and they will be returned on output in the MQSD structure. Only the following MQSD fields are used: StrucId, Version, Options, AlternateUserId and AlternateSecurityId, and SubName.

The call fails with reason code MQRC_NO_SUBSCRIPTION if a subscription does not exist matching the full subscription name. This failure can be avoided by combining the MQSO_CREATE option with MQSO_RESUME.

The userid of the subscription is the userid that created the subscription, or if it has been subsequently altered by a different userid, it is the userid of the most recent successful alteration. If an AlternateUserId is used, and use of alternate user IDs is allowed for that user, the alternate userid will be recorded as the userid that created the subscription instead of the userid under which the subscription was made.

If a matching subscription exists that was created without the MQSO_ANY_USERID option, and the userid of the subscription is different from that of the application requesting a handle to the subscription, the call fails with reason code MQRC_IDENTITY_MISMATCH.

If a matching subscription exists and is currently in use by another application, the call fails with MQRC_SUBSCRIPTION_IN_USE. If it is currently in use by the same connection the call will not fail and a handle to the subscription will be returned.

If the subscription named in *SubName* is not a valid subscription to resume or alter from an application, the call will fail with MQRC_INVALID_SUBSCRIPTION.

MQSO_RESUME is implied by MQSO_ALTER so you do not need to combine it with that option. However, combining the two options does not cause an error.

MQSO_ALTER

Return a handle to a pre-existing subscription with the full subscription name matching that specified by the name in *SubName*. Any attributes of the subscription that are different from that specified in the MQSD are altered in the subscription unless alteration is disallowed for that attribute. Details are noted in the description of each attribute and are summarized in the table below. If you try to alter an attribute that can not be changed, or to alter a subscription that has set the MQSO_IMMUTABLE option, the call fails with the reason code shown in the table below.

The call fails with reason code MQRC_NO_SUBSCRIPTION if a subscription matching the full subscription name does not exist. You can avoid this failure by combining the MQSO_CREATE option with MQSO_ALTER.

Combining MQSO_CREATE with MQSO_ALTER returns a handle to a pre-existing subscription for the specified *SubName* if one is found; if there is no existing subscription, a new one is created using all the fields provided in the MQSD.

The userid of the subscription is the userid that created the subscription, or if it has been subsequently altered by a different userid, it is the userid of the most recent, successful alteration. If an AlternateUserId is used, and use of alternate user IDs is allowed for that user, then the alternate userid will be recorded as the userid that created the subscription instead of the userid under which the subscription was made.

If a matching subscription exists that was created without the option MQSO_ANY_USERID and the userid of the subscription is different from that of the application requesting a handle to the subscription, the call fails with reason code MQRC_IDENTITY_MISMATCH.

If a matching subscription exists and is currently in use by another application, the call fails with MQRC_SUBSCRIPTION_IN_USE. If it is currently in use by the same connection the call will not fail and a handle to the subscription will be returned.

If the subscription named in *SubName* is not a valid subscription to resume or alter from an application, the call will fail with MQRC_INVALID_SUBSCRIPTION.

The following table shows the ability of MQSO_ALTER to alter attribute values in MQSD and MQSUB.

Data type descriptor or function call	Field name	Can this attribute be altered using MQSO_ALTER	Reason Code
MQSD	Durability options	No	MQRC_DURABILITY_NOT_ALTERABLE
MQSD	Destination Options	Yes	None
MQSD	Registration options	Yes (see note 1 on page 340)	MQRC_GROUPING_NOT_ALTERABLE if you try to alter MQSO_GROUP_SUB
MQSD	Publication options	Yes (see note 2 on page 340)	None
MQSD	Wildcard options	No	MQRC_TOPIC_NOT_ALTERABLE
MQSD	Other options	No (see note 3 on page 340)	None

Data type descriptor or function call	Field name	Can this attribute be altered using MQSO ALTER	Reason Code
MQSD	ObjectName	No	MQRC_TOPIC_NOT_ALTERABLE
MQSD	AlternateUserId	No (see note 4)	None
MQSD	AlternateSecurityId	No (see note 4)	None
MQSD	SubExpiry	Yes	None
MQSD	ObjectString	No	MQRC_TOPIC_NOT_ALTERABLE
MQSD	SubName	No (see note 5)	None
MQSD	SubUserData	Yes	None
MQSD	SubCorrelId	Yes (see note 6)	MQRC_GROUPING_NOT_ALTERABLE when in a grouped subscription
MQSD	PubPriority	Yes	None
MQSD	PubAccountingToken	Yes	None
MQSD	PubAppIdentityData	Yes	None
MQSD	SubLevel	No	MQRC_SUBLEVEL_NOT_ALTERABLE
MQSUB	Hobj	Yes (see note 6)	MQRC_GROUPING_NOT_ALTERABLE when in a grouped subscription
Notes: <ol style="list-style-type: none"> 1. MQSO_GROUP_SUB cannot be altered. 2. MQSO_NEW_PUBLICATIONS_ONLY cannot be altered because it is not part of the subscription 3. These options are not part of the subscription 4. This attribute is not part of the subscription 5. This attribute is the identity of the subscription being altered 6. Alterable except when part of a grouped sub (MQSO_GROUP_SUB) 			

Durability options: The following options control how durable the subscription is. You can specify only one of these options. If you are altering an existing subscription using the MQSO ALTER option, you cannot change the durability of the subscription. On return from an MQSUB call using MQSO RESUME the appropriate durability option is set.

MQSO_DURABLE

Request that the subscription to this topic remains until it is explicitly removed using MQCLOSE with the MQCO_REMOVE_SUB option. If this subscription is not explicitly removed it will remain even after this application's connection to the queue manager is closed.

If a durable subscription is requested to a topic that is defined as not allowing durable subscriptions, the call fails with MQRC_DURABILITY_NOT_ALLOWED.

MQSO_NON_DURABLE

Request that the subscription to this topic is removed when the application's connection to the queue manager is closed, if it has not already been explicitly removed. MQSO_NON_DURABLE is the opposite of the MQSO_DURABLE option, and is defined to aid program documentation. It is the default if neither is specified.

Destination options: The following option controls the destination that publications for a topic that has been subscribed to are sent to. If altering an existing subscription using the MQSO ALTER option, the destination used for publications for the subscription can be changed. On return from an MQSUB call using MQSO RESUME this option will set if appropriate.

MQSO_MANAGED

Request that the destination that the publications are sent to is managed by the queue manager.

The object handle returned in *Hobj* represents a queue manager managed queue and is for use with subsequent MQGET, MQCB, MQINQ, or MQCLOSE calls.

An object handle returned from a previous MQSUB call cannot be provided in the *Hobj* parameter when MQSO_MANAGED is not specified.

Scope Option: The following option controls the scope of the subscription being made. If altering an existing subscription using the MQSO_ALTER option, this subscription scope option cannot be changed. On returning from an MQSUB call using MQSO-RESUME, the appropriate scope option will be set.

MQSO_SCOPE_QMGR

This subscription is made only on the local queue manager. No proxy subscription is distributed to other queue managers in the network. Only publications that are published at this queue manager are sent to this subscriber. This overrides any behavior set using the SUBSCOPE topic attribute.

Note: If not set, the subscription scope is determined by the SUBSCOPE topic attribute.

Registration options: The following options control the details of the registration that is made to the queue manager for this subscription. If altering an existing subscription using the MQSO_ALTER option, these registration options can be changed. On return from an MQSUB call using MQSO-RESUME the appropriate registration options will be set.

MQSO_GROUP_SUB

This subscription is to be grouped with other subscriptions of the same SubLevel using the same queue and specifying the same correlation ID so that any publications to topics that would cause more than one publication message to be provided to the group of subscriptions, due to an overlapping set of topic strings being used, only causes one message to be delivered to the queue. If this option is not used, then each unique subscription (identified by SubName) that matches is provided with a copy of the publication which could mean more than one copy of the publication may be placed on the queue shared by a number of subscriptions.

Only the most significant subscription in the group is provided with a copy of the publication. The most significant subscription is based on the Full topic name up to the point where a wildcard is found. If a mixture of wildcard schemes is used within the group, only the position of the wildcard is important. You are advised not to combine different wildcard schemes within a group of subscriptions that share the same queue.

When creating a new grouped subscription it must still have a unique SubName, but if it matches the full topic name of an existing subscription in the group, the call will fail with MQRC_DUPLICATE_GROUP_SUB.

If the most significant subscription in group also specifies MQSO_NOT_OWN_PUBS and this is a publication from the same application, then no publication is delivered to the queue.

When altering a subscription made with this option, the fields which imply the grouping, Hobj on the MQSUB call (representing the queue and queue manager name), and the SubCorrelId cannot be changed. Attempting to alter them will cause the call will fail with MQRC_GROUPING_NOT_ALTERABLE.

This option must be combined with MQSO_SET_CORREL_ID with a SubCorrelId that is not set to MQCI_NONE, and cannot be combined with MQSO_MANAGED.

MQSO_ANY_USERID

When MQSO_ANY_USERID is specified, the identity of the subscriber is not restricted to a single userid. This allows any user to alter or resume the subscription when they have suitable authority. Only a single user may have the subscription at any one time. An attempt to resume use of a subscription currently in use by another application will cause the call to fail with MQRC_SUBSCRIPTION_IN_USE.

To add this option to an existing subscription the MQSUB call (using MQSO_ALTER) must come from the same userid as the original subscription itself.

If an MQSUB call refers to an existing subscription with MQSO_ANY_USERID set, and the userid differs from the original subscription, the call succeeds only if the new userid has authority to subscribe to the topic. On successful completion, future publications to this subscriber are put to the subscriber's queue with the new userid set in the publication message.

Do not specify both MQSO_ANY_USERID and MQSO_FIXED_USERID. If neither is specified, the default is MQSO_FIXED_USERID.

MQSO_FIXED_USERID

When MQSO_FIXED_USERID is specified, the subscription can be altered or resumed by only the last userid to alter the subscription. If the subscription has not been altered, it is the userid that created the subscription.

If an MQSUB verb refers to an existing subscription with MQSO_ANY_USERID set and alters the subscription using MQSO_ALTER to use option MQSO_FIXED_USERID, the userid of the subscription is now fixed at this new user id. The call succeeds only if the new userid has authority to subscribe to the topic.

If a user id other than the one recorded as owning a subscription tries to resume or alter an MQSO_FIXED_USERID subscription, the call fails with MQRC_IDENTITY_MISMATCH. The owning user id of a subscription can be viewed using the DISPLAY SBSTATUS command.

Do not specify both MQSO_ANY_USERID and MQSO_FIXED_USERID. If neither is specified, the default is MQSO_FIXED_USERID.

Publication options: The following options control the way publications are sent to this subscriber. If altering an existing subscription using the MQSO_ALTER option, these publication options can be changed.

MQSO_NOT_OWN_PUBS

Tells the broker that the application does not want to see any of its own publications. Publications are considered to have originated from the same

application if the connection handles are the same. On return from an MQSUB call using MQSO_RESUME this option will be set if appropriate.

MQSO_NEW_PUBLICATIONS_ONLY

No currently retained publications are to be sent, when this subscription is created, only new publications. This option only applies when MQSO_CREATE is specified. Any subsequent changes to a subscription do not alter the flow of publications and so any publications that have been retained on a topic, will have already been sent to the subscriber as new publications.

If this option is specified without MQSO_CREATE the call fails with MQRC_OPTIONS_ERROR. On return from an MQSUB call using MQSO_RESUME this option will not be set even if the subscription was created using this option.

If this option is not used, previously retained messages will be sent to the destination queue provided. If this action fails due to an error, either MQRC_RETAINED_MSG_Q_ERROR or MQRC_RETAINED_NOT_DELIVERED, the creation of the subscription will fail.

MQSO_PUBLICATIONS_ON_REQUEST

Setting this option indicates that the subscriber will request information specifically when required. The queue manager will not to send unsolicited messages to the subscriber. The retained publication (or possibly multiple publications if a wildcard is specified in the topic) will be sent to the subscriber each time a MQSUBRQ call is made using the Hsub handle from a previous MQSUB call. No publications will be sent as a result of the MQSUB call using this option. On return from an MQSUB call using MQSO_RESUME this option will be set if appropriate.

This option is not valid in combination with a SubLevel greater than 1.

Wildcard options: The following options control how wildcards are interpreted in the string provided in the ObjectString field of the MQSD. You can specify only one of these options. If altering an existing subscription using the MQSO_ALTER option, these wildcard options cannot be changed. On return from an MQSUB call using MQSO_RESUME the appropriate wildcard option will be set.

MQSO_WILDCARD_CHAR

Wildcards only operate on characters within the topic string.

The behavior defined by MQSO_WILDCARD_CHAR is shown in the table below.

Special Character	Behaviour
/	No significance, just another character
*	Wildcard, zero or more characters
?	Wildcard, one character
%	Escape character to allow the characters '*', '?' or '%' to be used in a string and not be interpreted as a special character, for example, '%*', '%?' or '%%'.

For example, publishing on the following topic:

/level0/level1/level2/level3/level4

matches subscribers using the following topics:

```

*
/*
/ level0/level1/level2/level3/*
/ level0/level1*/level3/level4
/ level0/level1/level2/level3/level4

```

Note: This use of wildcards supplies exactly the meaning provided in WebSphere MQ V6 and WebSphere MB V6 when using MQRFH1 formatted messages for Publish/Subscribe. It is recommended that this is not used for newly written applications and is only used for applications that were previously running against that version and have not been changed to use the default wildcard behaviour as described in MQSO_WILDCARD_TOPIC.

MQSO_WILDCARD_TOPIC

Wildcards only operate on topic elements within the topic string. This is the default behavior if none is chosen.

The behavior required by MQSO_WILDCARD_TOPIC is shown in the following table:

Special Character	Behaviour
/	Topic level separator
#	Wildcard: multiple topic level
+	Wildcard: single topic level
Notes:	
The '+' and '#' are not treated as wildcards if they are mixed in with other characters (including themselves) within a topic level. In the following string, the '#' and '+' characters are treated as ordinary characters. level0/level1/#+/level3/level#	

For example, publishing on the following topic:

```
/level0/level1/level2/level3/level4
```

matches subscribers using the following topics:

```

#
/#
/ level0/level1/level2/level3/#
/ level0/level1+/level3/level4

```

Note: This use of wildcards supplies the meaning provided in WebSphere Message Brokers Version 6 when using MQRFH2 formatted messages for Publish/Subscribe.

Other options: The following options control the way the API call is issued rather than the subscription. On return from an MQSUB call using MQSO_RESUME these options will be unchanged. See "AlternateUserId" on page 335 for more details.

MQSO_ALTERNATE_USER_AUTHORITY

The AlternateUserId field contains a user identifier to use to validate this MQSUB call. The call can succeed only if this AlternateUserId is authorized to open the object with the specified access options, regardless of whether the user identifier under which the application is running is authorized to do so.

MQSO_SET_CORRELID

The subscription is to use the correlation identifier supplied in the *SubCorrelId* field. If this option is not specified, a correlation identifier will be automatically created by the queue manager at subscription time and will be returned to the application in the *SubCorrelId* field. See “SubCorrelId (MQBYTE24)” on page 348 for more information.

This option cannot be combined with MQSO_MANAGED.

MQSO_SET_IDENTITY_CONTEXT

The subscription is to use the accounting token and application identity data supplied in the *PubAccountingToken* and *PubApplIdentityData* fields.

If this option is specified, the same authorization check is carried out as if the destination queue was accessed using an MQOPEN call with MQOO_SET_IDENTITY_CONTEXT, except in the case where the MQSO_MANAGED option is also used in which case there is no authorization check on the destination queue.

If this option is not specified, the publications sent to this subscriber will have default context information associated with them as follows:

Field in MQMD	Value used
<i>UserIdentifier</i>	The user id associated with the subscription at the time the subscription was made.
<i>AccountingToken</i>	Determined from the environment if possible; Set to MQACT_NONE if not.
<i>ApplIdentityData</i>	Set to blanks

This option is only valid with MQSO_CREATE and MQSO_ALTER. If used with MQSO_RESUME, the *PubAccountingToken* and *PubApplIdentityData* fields are ignored, so this option has no effect.

If a subscription is altered without using this option where previously the subscription had supplied identity context information, default context information will be generated for the altered subscription.

If a subscription allowing different user ids to use it with option MQSO_ANY_USERID, is resumed by a different user id, default identity context will be generated for the new user id now owning the subscription and any subsequent publications will be delivered containing the new identity context.

MQSO_FAIL_IF QUIESCING

The MQSUB call fails if the queue manager is in quiescing state. On z/OS, for a CICS or IMS application, this option also forces the MQSUB call to fail if the connection is in quiescing state.

PubAccountingToken (MQBYTE32)

This is the value that will be in the *AccountingToken* field of the Message Descriptor (MQMD) of all publication messages matching this subscription. *AccountingToken* is part of the identity context of the message. For more information about message context, see the *Application Programming Guide*. For more information about the *AccountingToken* field in the MQMD, see “AccountingToken (MQBYTE32)” on page 183

You can use the following special value for the *PubAccountingToken* field:

MQACT_NONE

No accounting token is specified.

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

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

If the option MQSO_SET_IDENTITY_CONTEXT is not specified, the accounting token is generated by the queue manager as default context information and this field is an output field which contains the *AccountingToken* which will be set in each message published for this subscription.

If the option MQSO_SET_IDENTITY_CONTEXT is specified, the accounting token is being generated by the user and this field is an input field which contains the *AccountingToken* to be set in each publication for this subscription.

The length of this field is given by MQ_ACCOUNTING_TOKEN_LENGTH. The initial value of this field is MQACT_NONE.

If altering an existing subscription using the MQSO_ALTER option, the value of *AccountingToken* in any future publication messages can be changed.

On return from an MQSUB call using MQSO_RESUME, this field is set to the current *AccountingToken* being used for the subscription.

PubApplIdentityData (MQCHAR32)

This is the value that will be in the *ApplIdentityData* field of the Message Descriptor (MQMD) of all publication messages matching this subscription. *ApplIdentityData* is part of the identity context of the message. For more information about message context, see Message context. For more information about the *ApplIdentityData* field in the MQMD, see “*ApplIdentityData (MQCHAR32)*” on page 185

If the option MQSO_SET_IDENTITY_CONTEXT is not specified, the *ApplIdentityData* which will be set in each message published for this subscription is blanks, as default context information.

If the option MQSO_SET_IDENTITY_CONTEXT is specified, the *PubApplIdentityData* is being generated by the user and this field is an input field which contains the *ApplIdentityData* to be set in each publication for this subscription.

The length of this field is given by MQ_APPL_IDENTITY_DATA_LENGTH. The initial value of this field is the null string in C, and 32 blank characters in other programming languages.

If altering an existing subscription using the MQSO_ALTER option, the *ApplIdentityData* of any future publication messages can be changed.

On return from an MQSUB call using MQSO_RESUME, this field is set to the current *ApplIdentityData* being used for the subscription.

PubPriority (MQLONG)

This is the value that will be in the *Priority* field of the Message Descriptor (MQMD) of all publication messages matching this subscription. For more information about the *Priority* field in the MQMD, see “Priority (MQLONG)” on page 214.

The value must be greater than or equal to zero; zero is the lowest priority. The following special values can also be used:

MQPRI_PRIORITY_AS_Q_DEF

When a subscription queue is provided in the *Hobj* field in the MQSUB call, and is not a managed handle, then the priority for the message is taken from the *DefPriority* attribute of this queue. If the queue is a cluster queue or there is more than one definition in the queue-name resolution path then the priority is determined when the publication message is put to the queue as described for “Priority (MQLONG)” on page 214.

If the MQSUB call uses a managed handle, the priority for the message is taken from the *DefPriority* attribute of the model queue associated with the topic subscribed to.

MQPRI_PRIORITY_AS_PUBLISHED

The priority for the message is the priority of the original publication. This is the initial value of the field.

If altering an existing subscription using the MQSO_ALTER option, the *Priority* of any future publication messages can be changed.

On return from an MQSUB call using MQSO_RESUME, this field is set to the current priority being used for the subscription.

ResObjectString (MQCHARV)

This is the long object name after the queue manager resolves the name provided in *ObjectName*.

If the long object name is provided in *ObjectString* and nothing is provided in *ObjectName*, then the value returned in this field is the same as provided in *ObjectString*.

If this field is omitted (that is *ResObjectString.VSBufSize* is zero) then the *ResObjectString* will not be returned, but the length will be returned in *ResObjectString.VSLength*. If the length is shorter than the full *ResObjectString* then it will be truncated and will return as many of the rightmost characters as can fit in the provided length.

If *ResObjectString* is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code MQRC_RES_OBJECT_STRING_ERROR.

SelectionString (MQCHARV)

This is the string used to provide the selection criteria used when subscribing for messages from a topic.

This variable length field will be returned on output from an MQSUB call using the MQSO_RESUME option, if a buffer is provided, and also there is a positive buffer length in *VSBufSize*. If no buffer is provided on the call, only the length of

| the selection string will be returned in the VSLength field of the MQCHARV. If the
| buffer provided is smaller than the space required to return the field, only
| VSBufSize bytes are returned in the provided buffer.

| If *SelectionString* is specified incorrectly, according to the description of how to
| use the MQCHARV structure, or if it exceeds the maximum length, the call fails
| with reason code MQRC_SELECTION_STRING_ERROR.

StrucId (MQCHAR4)

This is the structure identifier; the value must be:

MQSD_STRUC_ID

Identifier for Subscription Descriptor structure.

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

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

SubCorrelId (MQBYTE24)

This field contains a correlation identifier common to all publications matching this subscription.

| **Attention:** a correlation identifier can only be passed between queue managers in a
| publish/subscribe cluster, not a hierarchy.

All publications sent to match this subscription contain this correlation identifier in the message descriptor. If multiple subscriptions get their publications from the same queue, using MQGET by correlation identifier allows only publications for a specific subscription to be obtained. This correlation identifier can either be generated by the queue manager or by the user.

| If the option MQSO_SET_CORREL_ID is not specified, the correlation identifier is
| generated by the queue manager and this field is an output field containing the
| correlation identifier that will be set in each message published for this
| subscription. The generated correlation identifier consists of a 4-byte product
| identifier (AMQX or CSQM in either ASCII or EBCDIC) followed by a product
| specific implementation of a unique string.

If the option MQSO_SET_CORREL_ID is specified, the correlation identifier is generated by the user and this field is an input field containing the correlation identifier to be set in each publication for this subscription. In this case, if the field contains MQCI_NONE, the correlation identifier that is set in each message published for this subscription is the correlation identifier created by the original put of the message.

| If the option MQSO_GROUP_SUB is specified and the correlation identifier
| specified is the same as an existing grouped subscription using the same queue
| and an overlapping topic string, only the most significant subscription in the group
| is provided with a copy of the publication.

The length of this field is given by MQ_CORREL_ID_LENGTH. The initial value of this field is MQCI_NONE.

If you are altering an existing subscription using the MQSO_ALTER option, and this field is an input field, then the subscription correlation identifier can be changed, unless the subscription is a grouped subscription, that is, it has been created using the option MQSO_GROUP_SUB, in which case the subscription correlation identifier cannot be changed.

On return from an MQSUB call using MQSO_RESUME, this field is set to the current correlation identifier for the subscription.

SubExpiry (MQLONG)

This is the period of time expressed in tenths of a second after which the subscription expires. No more publications will match this subscription after this interval has passed. As soon as a subscription expires publications are no longer sent to the queue, however the publications that are already there are not affected in any way. *SubExpiry* has NO EFFECT on publication expiry.

The following special value is recognized:

MQEI_UNLIMITED

The subscription has an unlimited expiration time.

If altering an existing subscription using the MQSO_ALTER option, the expiry of the subscription can be changed.

On return from an MQSUB call using the MQSO_RESUME option this field will be set to the original expiry of the subscription and not the remaining expiry time.

SubLevel (MQLONG)

This is the level associated with the subscription. Publications will only be delivered to this subscription if it is in the set of subscriptions with the highest SubLevel value less than or equal to the PubLevel used at publication time.

The value must be in the range zero to 9. Zero is the lowest level.

The initial value of this field is 1.

For more information see "Intercepting publications" in the *WebSphere MQ Publish/Subscribe User's Guide*.

If altering an existing subscription using the MQSO_ALTER option, then the SubLevel cannot be changed.

Combining a SubLevel with a value greater than 1 with the option MQSO_PUBLICATIONS_ON_REQUEST is not allowed.

On return from an MQSUB call using MQSO_RESUME, this field is set to the current level being used for the subscription.

SubUserData (MQCHARV)

This specifies the subscription user data. The data provided on the subscription in this field will be included as the MQSubUserData message property of every publication sent to this subscription.

The maximum length of *SubUserData* is 10240.

If *SubUserData* is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code MQRC_SUB_USER_DATA_ERROR.

This is an input field. The initial values of the fields in this structure are the same as those in the MQCHARV structure.

If altering an existing subscription using the MQSO_ALTER option, the subscription user data can be changed.

This variable length field is returned on output from an MQSUB call using the MQSO_RESUME option, if a buffer is provided and there is a positive buffer length in *VSBufLen*. If no buffer is provided on the call, only the length of the subscription user data is returned in the *VSLength* field of the MQCHARV. If the buffer provided is smaller than the space required to return the field, only *VSBufLen* bytes are returned in the provided buffer.

SubName (MQCHARV)

This specifies the subscription name. This field is only required if *Options* specifies the option MQSO_DURABLE, but if provided will be used by the queue manager for MQSO_NON_DURABLE as well.

If specified, *SubName* must be unique within the queue manager, because it is the method used to identify the subscription.

The maximum length of *SubName* is 10240.

This field serves two purposes. For an MQSO_DURABLE subscription, you use this field to identify a subscription so you can resume it after it has been created if you have either closed the handle to the subscription (using the MQCO_KEEP_SUB option) or have been disconnected from the queue manager. This is done using the MQSUB call with the MQSO_RESUME option. It is also displayed in the administrative view of subscriptions in the SUBNAME field in DISPLAY SBSTATUS.

If *SubName* is specified incorrectly, according to the description of how to use the MQCHARV structure, is left out when it is required (that is *SubName.VSLength* is zero), or if it exceeds the maximum length, the call fails with reason code MQRC_SUB_NAME_ERROR.

This is an input field. The initial values of the fields in this structure are the same as those in the MQCHARV structure.

If altering an existing subscription using the MQSO_ALTER option, the subscription name cannot be changed, because it is the identifying field used to find the referenced subscription. It is not changed on output from an MQSUB call with the MQSO_RESUME option.

Version (MQLONG)

This is the structure version number; the value must be:

MQSD_VERSION_1

Version-1 Subscription Descriptor structure.

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

MQSD_CURRENT_VERSION

Current version of Subscription Descriptor structure.

This is always an input field. The initial value of this field is MQSD_VERSION_1.

Using topic strings

The full topic name is given by the concatenation of two parts. A part exists if the first character of the field is neither a blank nor a null character:

1. The value of the TOPICSTR parameter of the topic object named in *ObjectName*
2. *ObjectString*, if the *VSLength* provided for that variable length string is non-zero

If one of these parts exists it is used unchanged as the topic name.

If neither part exists the call fails with reason code MQRC_TOPIC_STRING_ERROR.

If both parts exist, they are concatenated in the order they are listed above. A forward slash character (/) is inserted between them in the resultant combined topic if one is required.

The following table shows examples of topic string concatenation. A topic has been defined using the command DEF TOPIC(TOP1) TOPICSTR(/Football) and you specify TOP1 as the *ObjectName*.

TOPICSTR	ObjectString	Concatenation result	Comment
/Football	Scores	/Football/Scores	A forward slash is added at the concatenation point
/Football	/Scores	/Football/Scores	

Notes:

1. The forward slash character is considered to be a special character providing structure to the full topic name. If you use this character for any other reason the structure of the topic tree will not be as you expect. This means that the topic '/Football' is not the same as the topic 'Football'.
2. A full topic name with two repeated forward slash characters is not valid.
3. If the full topic name is not valid, the call fails with reason code MQRC_TOPIC_STRING_ERROR.
4. The following wildcard characters are special characters: + # * ?. They are not considered invalid, however you should take care to understand the behavior when using them. You might prefer not to use these characters in your topic strings when publishing.
 - Publishing on a topic string with # or + mixed in with other characters (including themselves) within a topic level can be subscribed on, with either wildcard scheme.
 - Publishing on a topic string with # or + as the only character between two '/' characters will produce a topic string that cannot be subscribed on explicitly by an application using the wildcard scheme MQSO_WILDCARD_TOPIC. This will result in the application getting more publications than expected.

Initial values and language declarations for MQSD

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQSD_STRUC_ID	'SDBb'
<i>Version</i>	MQSD_VERSION_1	1
<i>Options</i>	MQSO_NON_DURABLE	0
<i>ObjectName</i>	None	Null string or blanks
<i>AlternateUserId</i>	None	Null string or blanks
<i>AlternateSecurityId</i>	MQSID_NONE	Nulls
<i>SubExpiry</i>	MQEI_UNLIMITED	-1
<i>ObjectString</i>	None	Names and values as defined for MQCHARV
<i>SubName</i>	None	Names and values as defined for MQCHARV
<i>SubUserData</i>	None	Names and values as defined for MQCHARV
<i>SubCorrelId</i>	MQCI_NONE	Nulls
<i>PubPriority</i>	MQPRI_PRIORITY_AS_Q_DEF	-3
<i>PubAccountingToken</i>	MQACT_NONE	Nulls
<i>PubApplIdentityData</i>	None	Null string or blanks
<i>Selection String</i>	None	Names and values as defined for MQCHARV
<i>SubLevel</i>	None	1
<i>ResObjectString</i>	None	Names and values as defined for MQCHARV

Notes:

1. The symbol *b* represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQSD_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQSD MySD = {MQSD_DEFAULT};
```

C declaration

```
typedef struct tagMQSD MQSD;
struct tagMQSD {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQLONG    Options;          /* Options associated with subscribing */
    MQCHAR48  ObjectName;       /* Object name */
    MQCHAR12  AlternateUserId;   /* Alternate user identifier */
    MQBYTE40  AlternateSecurityId; /* Alternate security identifier */
    MQLONG    SubExpiry;        /* Expiry of Subscription */
    MQCHARV   ObjectString;     /* Object Long name */
    MQCHARV   SubName;          /* Subscription name */
    MQCHARV   SubUserData;      /* Subscription User data */
    MQBYTE24  SubCorrelId;      /* Correlation Id related to this subscription */
    MQLONG    PubPriority;       /* Priority set in publications */
};
```



```

MQBYTE32 PubAccountingToken; /* Accounting Token set in publications */
MQCHAR32 PubApplIdentityData; /* Appl Identity Data set in publications */
MQCHARV SelectionString; /* Message selector structure */
MQLONG SubLevel; /* Subscription level */
MQCHARV ResObjectString; /* Resolved Long object name*/
/* Ver:1 */
};

```

COBOL declaration

```

** Address of variable length string
20 MQSD-OBJECTSTRING-VSPTR POINTER.
** Offset of variable length string
20 MQSD-OBJECTSTRING-VSOFFSET PIC S9(9) BINARY.
** size of buffer
20 MQSD-OBJECTSTRING-VSBUFSIZE PIC S9(9) BINARY.
** Length of variable length string
20 MQSD-OBJECTSTRING-VSLENGTH PIC S9(9) BINARY.
** CCSID of variable length string
20 MQSD-OBJECTSTRING-VSCCSID PIC S9(9) BINARY.
** Subscription name
15 MQSD-SUBNAME.
** Address of variable length string
20 MQSD-SUBNAME-VSPTR POINTER.
** Offset of variable length string
20 MQSD-SUBNAME-VSOFFSET PIC S9(9) BINARY.
** size of buffer
20 MQSD-SUBNAME-VSBUFSIZE PIC S9(9) BINARY.
** Length of variable length string
20 MQSD-SUBNAME-VSLENGTH PIC S9(9) BINARY.
** CCSID of variable length string
20 MQSD-SUBNAME-VSCCSID PIC S9(9) BINARY.
** Subscription User data
15 MQSD-SUBUSERDATA.
** Address of variable length string
20 MQSD-SUBUSERDATA-VSPTR POINTER.
** Offset of variable length string
20 MQSD-SUBUSERDATA-VSOFFSET PIC S9(9) BINARY.
** size of buffer
20 MQSD-SUBUSERDATA-VSBUFSIZE PIC S9(9) BINARY.
** Length of variable length string
20 MQSD-SUBUSERDATA-VSLENGTH PIC S9(9) BINARY.
** CCSID of variable length string
20 MQSD-SUBUSERDATA-VSCCSID PIC S9(9) BINARY.
** Correlation Id related to this subscription
15 MQSD-SUBCORRELID PIC X(24).
** Priority set in publications
15 MQSD-PUBPRIORITY PIC S9(9) BINARY.
** Accounting Token set in publications
15 MQSD-PUBACCOUNTINGTOKEN PIC X(32).
** Appl Identity Data set in publications
15 MQSD-PUBAPPLIDENTITYDATA PIC X(32).
** Message Selector
15 MQSD-SELECTIONSTRING.
** Address of variable length string
20 MQSD-SELECTIONSTRING-VSPTR POINTER.
** Offset of variable length string
20 MQSD-SELECTIONSTRING-VSOFFSET PIC S9(9) BINARY.
** size of buffer
20 MQSD-SELECTIONSTRING-VSBUFSIZE PIC S9(9) BINARY.
** Length of variable length string
20 MQSD-SELECTIONSTRING-VSLENGTH PIC S9(9) BINARY.
** CCSID of variable length string
20 MQSD-SELECTIONSTRING-VSCCSID PIC S9(9) BINARY.
** Selection criteria
20 MQSD-SELECTIONSTRING-SUBLEVEL PIC S9(9) BINARY.
** Long object name
20 MQSD-SELECTIONSTRING-RESOBJSTRING PIC S9(9) BINARY.

```

PL/I declaration

```

dcl
1 MQSD based,
3 StrucId          char(4),          /* Structure identifier */
3 Version          fixed bin(31),   /* Structure version number */
3 Options          fixed bin(31),   /* Options associated with subscribing */
3 ObjectName       char(48),        /* Object name */
3 AlternateUserId  char(12),        /* Alternate user identifier */
3 AlternateSecurityId char(40),     /* Alternate security identifier */
3 SubExpiry        fixed bin(31),   /* Expiry of Subscription */
3 ObjectString,    /* Object Long name */
5 VSPtr           pointer,         /* Address of variable length string */
5 VSOffset        fixed bin(31),   /* Offset of variable length string */
5 VSBufSize       fixed bin(31),   /* size of buffer */
5 VSLength        fixed bin(31),   /* Length of variable length string */
5 VSCCSID         fixed bin(31);   /* CCSID of variable length string */
3 SubName,        /* Subscription name */
5 VSPtr           pointer,         /* Address of variable length string */
5 VSOffset        fixed bin(31),   /* Offset of variable length string */
5 VSBufSize       fixed bin(31),   /* size of buffer */
5 VSLength        fixed bin(31),   /* Length of variable length string */
5 VSCCSID         fixed bin(31);   /* CCSID of variable length string */
3 SubUserData,   /* Subscription User data */
5 VSPtr           pointer,         /* Address of variable length string */
5 VSOffset        fixed bin(31),   /* Offset of variable length string */
5 VSBufSize       fixed bin(31),   /* size of buffer */
5 VSLength        fixed bin(31),   /* Length of variable length string */
5 VSCCSID         fixed bin(31);   /* CCSID of variable length string */
3 SubCorrelId    char(24),         /* Correlation Id related to this subscription */
3 PubPriority     fixed bin(31),   /* Priority set in publications */
3 PubAccountingToken char(32),     /* Accounting Token set in publications */
3 PubApplIdentityData char(32),   /* Appl Identity Data set in publications */
3 SelectionString, /* Message Selection */
5 VSPtr           pointer,         /* Address of variable length string */
5 VSOffset        fixed bin(31),   /* Offset of variable length string */
5 VSBufSize       fixed bin(31),   /* size of buffer */
5 VSLength        fixed bin(31),   /* Length of variable length string */
5 VSCCSID         fixed bin(31);   /* CCSID of variable length string */
3 SubLevel       fixed bin(31),   /* Subscription level */
3 ResObjectString, /* Resolved Long object name */
5 VSPtr           pointer,         /* Address of variable length string */
5 VSOffset        fixed bin(31),   /* Offset of variable length string */
5 VSBufSize       fixed bin(31),   /* size of buffer */
5 VSLength        fixed bin(31),   /* Length of variable length string */
5 VSCCSID         fixed bin(31);   /* CCSID of variable length string */

```

System/390 assembler declaration

```

MQSD          DSECT
MQSD_STRUCID  DS      CL4   Structure identifier
MQSD_VERSION  DS      F     Structure version number
MQSD_OPTIONS  DS      F     Options associated with subscribing
MQSD_OBJECTNAME DS     CL48  Object name
MQSD_ALTERNATEUSERID DS    CL12  Alternate user identifier
MQSD_ALTERNATESECURITYID DS  CL40  Alternate security identifier
MQSD_SUBEXPIRY DS      F     Expiry of Subscription
MQSD_OBJECTSTRING DS     0F    Object Long name
MQSD_OBJECTSTRING_VSPTR DS    F     Address of variable length string
MQSD_OBJECTSTRING_VSOFFSET DS   F     Offset of variable length string
MQSD_OBJECTSTRING_VSBUFSIZE DS  F     size of buffer
MQSD_OBJECTSTRING_VSLLENGTH DS  F     Length of variable length string
MQSD_OBJECTSTRING_VSCCSID DS   F     CCSID of variable length string
MQSD_OBJECTSTRING_LENGTH EQU   *-MQSD_OBJECTSTRING
                                ORG   MQSD_OBJECTSTRING
MQSD_OBJECTSTRING_AREA DS     CL(MQSD_OBJECTSTRING_LENGTH)
*
MQSD_SUBNAME  DS     0F    Subscription name
MQSD_SUBNAME_VSPTR DS    F     Address of variable length string
MQSD_SUBNAME_VSOFFSET DS   F     Offset of variable length string
MQSD_SUBNAME_VSBUFSIZE DS  F     size of buffer
MQSD_SUBNAME_VSLLENGTH DS  F     Length of variable length string
MQSD_SUBNAME_VSCCSID DS   F     CCSID of variable length string
MQSD_SUBNAME_LENGTH EQU   *-MQSD_SUBNAME
                                ORG   MQSD_SUBNAME
MQSD_SUBNAME_AREA DS     CL(MQSD_SUBNAME_LENGTH)

```

```

| *
| MQSD_SUBUSERDATA          DS    0F    Subscription User data
| MQSD_SUBUSERDATA_VSPTR   DS    F     Address of variable length string
| MQSD_SUBUSERDATA_VSOFFSET DS    F     Offset of variable length string
| MQSD_SUBUSERDATA_VSBUFFSIZE DS    F     size of buffer
| MQSD_SUBUSERDATA_VSLENGTH DS    F     Length of variable length string
| MQSD_SUBUSERDATA_VSCCSID DS    F     CCSID of variable length string
| MQSD_SUBUSERDATA_LENGTH  EQU   *-MQSD_SUBUSERDATA
|                               ORG   MQSD_SUBUSERDATA
| MQSD_SUBUSERDATA_AREA    DS    CL(MQSD_SUBUSERDATA_LENGTH)
| *
| MQSD_SUBCORRELID         DS    CL24   Correlation Id related to this subscription
| MQSD_PUBPRIORITY         DS    F     Priority set in publications
| MQSD_PUBACCOUNTINGTOKEN DS    CL32   Accounting Token set in publications
| MQSD_PUBAPPLIDENTITYDATA DS    CL32   Appl Identity Data set in publications
| *
| MQSD_SELECTIONSTRING     DS    F     Message Selector
| MQSD_SELECTIONSTRING_VSPTR DS    F     Address of variable length string
| MQSD_SELECTIONSTRING_VSOFFSET DS    F     Offset of variable length string
| MQSD_SELECTIONSTRING_VSBUFFSIZE DS    F     size of buffer
| MQSD_SELECTIONSTRING_VSLENGTH DS    F     Length of variable length string
| MQSD_SELECTIONSTRING_VSCCSID DS    F     CCSID of variable length string
| MQSD_SELECTIONSTRING_LENGTH EQU   *- MQSD_SELECTIONSTRING
|                               ORG   MQSD_SELECTIONSTRING
| MQSD_SELECTIONSTRING_AREA DS    CL(MQSD_SELECTIONSTRING_LENGTH)
| *
| MQSD-SUBLEVEL            DS    F     Subscription level
| *
| MQSD_RESOBJECTSTRING     DS    F     Resolved Long object name
| MQSD_RESOBJECTSTRING_VSPTR DS    F     Address of variable length string
| MQSD_RESOBJECTSTRING_VSOFFSET DS    F     Offset of variable length string
| MQSD_RESOBJECTSTRING_VSBUFFSIZE DS    F     size of buffer
| MQSD_RESOBJECTSTRING_VSLENGTH DS    F     Length of variable length string
| MQSD_RESOBJECTSTRING_VSCCSID DS    F     CCSID of variable length string
| MQSD_RESOBJECTSTRING_LENGTH EQU   *- MQSD_RESOBJECTSTRING
|                               ORG   MQSD_RESOBJECTSTRING
| MQSD_RESOBJECTSTRING_AREA DS    CL(MQSD_RESOBJECTSTRING_LENGTH)
| *
| MQSD_LENGTH              EQU   *-MQSD
|                               ORG   MQSD
| MQSD_AREA                DS    CL(MQSD_LENGTH)

```

MQSMPO – Set message property options

The following table summarizes the fields in the structure.

Table 73. Fields in MQSMPO

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>Options</i>	Options	Options
<i>ValueEncoding</i>	Property value encoding	ValueEncoding
<i>ValueCCSID</i>	Property value character set	ValueCCSID

Overview for MQSMPO

Availability: All WebSphere MQ systems and WebSphere MQ clients.

Purpose: The MQSMPO structure allows applications to specify options that control how properties of messages are set. The structure is an input parameter on the MQSETMP call.

Character set and encoding: Data in MQSMPO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields for MQSMPO

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

Options (MQLONG)

Location options: The following options relate to the relative location of the property compared to the property cursor:

MQSMPO_SET_FIRST

Sets the value of the first property that matches the specified name, or if it does not exist, adds a new property after all other properties with a matching hierarchy.

MQSMPO_SET_PROP_UNDER_CURSOR

Sets the value of the property pointed to by the property cursor. The property pointed to by the property cursor is the one that was last inquired using either the MQIMPO_INQ_FIRST or the MQIMPO_INQ_NEXT option.

The property cursor is reset when the message handle is reused, or when the message handle is specified in the *MsgHandle* field of the MQGMO or MQPMO structure on an MQGET or MQPUT call respectively.

If this option is used when the property cursor has not yet been established or if the property pointer to by the property cursor has been deleted, the call fails with completion code MQCC_FAILED and reason code MQRC_PROPERTY_NOT_AVAILABLE.

MQSMPO_SET_PROP_BEFORE_CURSOR

Sets a new property before the property pointed to by the property cursor. The property pointed to by the property cursor is the one that was last inquired using either the MQIMPO_INQ_FIRST or the MQIMPO_INQ_NEXT option.

The property cursor is reset when the message handle is reused, or when the message handle is specified in the *MsgHandle* field of the MQGMO or MQPMO structure on an MQGET or MQPUT call respectively.

If this option is used when the property cursor has not yet been established or if the property pointer to by the property cursor has been deleted, the call fails with completion code MQCC_FAILED and reason code MQRC_PROPERTY_NOT_AVAILABLE.

MQSMPO_SET_PROP_AFTER_CURSOR

Sets a new property after the property pointed to by the property cursor. The property pointed to by the property cursor is the one that was last inquired using either the MQIMPO_INQ_FIRST or the MQIMPO_INQ_NEXT option.

The property cursor is reset when the message handle is reused, or when the message handle is specified in the *MsgHandle* field of the MQGMO or MQPMO structure on an MQGET or MQPUT call respectively.

If this option is used when the property cursor has not yet been established or if the property pointer to by the property cursor has been

deleted, the call fails with completion code MQCC_FAILED and reason code MQRC_PROPERTY_NOT_AVAILABLE.

If you need none of the options described, use the following option:

MQSMPO_NONE

No options specified.

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

StrucId (MQCHAR4)

This is the structure identifier; the value must be:

MQSMPO_STRUC_ID

Identifier for set message property options structure.

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

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

ValueCCSID (MQLONG)

The character set of the property value to be set if the value is a character string.

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

ValueEncoding (MQLONG)

The encoding of the property value to be set if the value is numeric.

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

Version (MQLONG)

This is the structure version number; the value must be:

MQSMPO_VERSION_1

Version-1 set message property options structure.

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

MQSMPO_CURRENT_VERSION

Current version of set message property options structure.

This is always an input field. The initial value of this field is MQSMPO_VERSION_1.

Initial values and language declarations for MQSMPO

Table 74. Initial values of fields in MQSMPO

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQSMPO_STRUC_ID	'SMPO'
<i>Version</i>	MQSMPO_VERSION_1	1

Table 74. Initial values of fields in MQSMPO (continued)

Field name	Name of constant	Value of constant
<i>Options</i>	MQSMPO_NONE	0
<i>ValueEncoding</i>	MQENC_NATIVE	Depends on environment
<i>ValueCCSID</i>	MQCCSI_APPL	-3
Notes: <ol style="list-style-type: none"> 1. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages. 2. In the C programming language, the macro variable MQSMPO_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure: <pre>MQSMPO MySMPO = {MQSMPO_DEFAULT};</pre> 		

C declaration

```
typedef struct tagMQSMPO MQSMPO;
struct tagMQSMPO {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQLONG    Options;          /* Options that control the action of MQSETMP */
    MQLONG    ValueEncoding;    /* Encoding of Value */
    MQLONG    ValueCCSID;       /* Character set identifier of Value */
};
```

COBOL declaration

```
** MQSMPO structure
10 MQSMPO.
** Structure identifier
15 MQSMPO-STRUCID PIC X(4).
** Structure version number
15 MQSMPO-VERSION PIC S9(9) BINARY.
** Options that control the action of MQSETMP
15 MQSMPO-OPTIONS PIC S9(9) BINARY.
** Encoding of VALUE
15 MQSMPO-VALUEENCODING PIC S9(9) BINARY.
** Character set identifier of VALUE
15 MQSMPO-VALUECCSID PIC S9(9) BINARY.
```

PL/I declaration

```
dc1
1 MQSMPO based,
3 StrucId char(4), /* Structure identifier */
3 Version fixed bin(31), /* Structure version number */
3 Options fixed bin(31), /* Options that control the action of MQSETMP */
3 ValueEncoding fixed bin(31), /* Encoding of Value */
3 ValueCCSID fixed bin(31), /* Character set identifier of Value */
```

System/390 assembler declaration

```
MQSMPO DSECT
MQSMPO_STRUCID DS CL4 Structure identifier
MQSMPO_VERSION DS F Structure version number
MQSMPO_OPTIONS DS F Options that control the action of
* MQSETMP
MQSMPO_VALUEENCODING DS F Encoding of VALUE
MQSMPO_VALUECCSID DS F Character set identifier of VALUE
MQSMPO_LENGTH EQU *-MQSMPO
MQSMPO_AREA DS CL(MQSMPO_LENGTH)
```

MQSRO - Subscription request options

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>Options</i>	Options	Options
<i>NumPubs</i>	Number of publications	NumPubs

Overview for MQSRO

Availability: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, z/OS plus WebSphere MQ clients connected to these systems.

Purpose: The MQSRO structure allows the application to specify options that control how a subscription request is made. The structure is an input/output parameter on the MQSUBRQ call.

Version: The current version of MQSRO is MQSRO_VERSION_1.

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

Fields for MQSRO

The MQSRO structure contains the following fields; the fields are described in alphabetical order:

NumPubs (MQLONG)

This is an output field, returned to the application to indicate the number of publications sent to the subscription queue as a result of this call. Although this number of publications have been sent as a result of this call, there is no guarantee that this many messages will be available for the application to get, especially if they are non-persistent messages.

There may be more than one publication if the topic subscribed to contained a wildcard. If no wildcards were present in the topic string when the subscription represented by *Hsub* was created, then at most one publication is sent as a result of this call.

Options (MQLONG)

One of the following options must be specified. Only one option can be specified.

MQSRO_FAIL_IF_QUIESCING

The MQSUBRQ call fails if the queue manager is in the quiescing state. On z/OS, for a CICS or IMS application, this option also forces the MQSUBRQ call to fail if the connection is in a quiescing state.

Default option: If the option described above is not required, the following option must be used:

MQSRO_NONE

Use this value to indicate that no other options have been specified; all options assume their default values.

MQSRO_NONE helps program documentation. Although it is not intended that this option be used with any other, because its value is zero, this use cannot be detected.

StrucId (MQCHAR4)

This is the structure identifier; the value must be:

MQSRO_STRUC_ID

Identifier for Subscription Request Options structure.

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

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

Version (MQLONG)

This is the structure version number; the value must be:

MQSRO_VERSION_1

Version-1 Subscription Request Options structure.

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

MQSRO_CURRENT_VERSION

Current version of Subscription Request Options structure.

This is always an input field. The initial value of this field is MQSRO_VERSION_1.

Initial values and language declarations for MQSRO

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQSRO_STRUC_ID	'SR0b'
<i>Version</i>	MQSRO_VERSION_1	1
<i>Options</i>	MQSRO_NONE	0
<i>NumPubs</i>	None	0

Notes:

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

```
MQSRO MySRO = {MQSRO_DEFAULT};
```

C declaration

```
typedef struct tagMQSRO MQSRO;
struct tagMQSRO {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQLONG    Options;          /* Options that control the action of MQSUBRQ */
    MQLONG    NumPubs;          /* Number of publications sent */
    /* Ver:1 */
};
```

COBOL declaration

```
** MQSRO structure
10 MQSRO.
** Structure identifier
15 MQSRO-STRUCID          PIC X(4).
** Structure version number
15 MQSRO-VERSION          PIC S9(9) BINARY.
** Options that control the action of MQSUBRQ
15 MQSRO-OPTIONS          PIC S9(9) BINARY.
** Number of publications sent
15 MQSRO-NUMPUBS          PIC S9(9) BINARY.
```

PL/I declaration

```
dc1
1 MQSRO based,
3 StrucId      char(4),      /* Structure identifier */
3 Version      fixed bin(31), /* Structure version number */
3 Options      fixed bin(31), /* Options that control the action of MQSUBRQ */
3 NumPubs      fixed bin(31); /* Number of publications sent */
```

System/390 assembler declaration

```
MQSRO          DSECT
MQSRO_STRUCID  DS   CL4   Structure identifier
MQSRO_VERSION  DS   F     Structure version number
MQSRO_OPTIONS  DS   F     Options that control the action of MQSUBRQ
MQSRO_NUMPUBS  DS   F     Number of publications sent
*
MQSRO_LENGTH   EQU   *-MQSRO
               ORG   MQSRO
MQSRO_AREA     DS   CL(MQSRO_LENGTH)
```

MQSTS – Status reporting structure

The following table summarizes the fields in the structure.

Table 75. Fields in MQSTS

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>CompCode</i>	Completion code of first error	CompCode
<i>Reason</i>	Reason code of first error	Reason
<i>PutSuccessCount</i>	Number of successful asynchronous put calls	SuccessCount
<i>PutWarningcount</i>	Number of asynchronous put calls which had warnings	WarningCount
<i>PutFailureCount</i>	Number of failed asynchronous put calls	FailureCount
<i>ObjectType</i>	Type of failing object	ObjectType

Table 75. Fields in MQSTS (continued)

Field	Description	Topic
<i>ObjectName</i>	Name of failing object	ObjectName
<i>ObjectQmgrName</i>	Name of queue manager owning the failing object	ObjectQmgrName
<i>ResolvedObjectName</i>	Resolved name of destination queue	ResolvedObjectName
<i>ResolvedQmgrName</i>	Resolved name of destination queue manager	ResolvedQmgrName

Overview for MQSTS

Purpose: The MQSTS structure is an output parameter from the MQSTAT command.

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

Usage: The MQSTAT command is used to retrieve status information. This information is returned in an MQSTS structure. For information about MQSTAT, see “MQSTAT – Retrieve status information” on page 561.

Fields for MQSTS

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

CompCode (MQLONG)

This is the completion code resulting from a previous asynchronous put operation on the object specified in *ObjectName*.

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

PutFailureCount (MQLONG)

This is a count of the number of asynchronous put operations that completed with a completion code of MQCC_FAILED.

This is always an output field. The initial value of this field is 0.

ObjectName (MQCHAR48)

This is the name of the object used in the put operation, the failure of which is reported in the *CompCode* and *Reason* fields.

This is always an output field. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ObjectQMgrName (MQCHAR48)

This is the name of the queue manager on which the *ObjectName* object is defined. A name that is entirely blank up to the first null character or the end of the field denotes the queue manager to which the application is connected (the local queue manager).

This is always an output field. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ObjectType (MQLONG)

The type of object being named in *ObjectName*. Possible values are:

MQOT_ALIAS_Q
Alias queue.

MQOT_LOCAL_Q
Local queue.

MQOT_MODEL_Q
Model queue.

MQOT_Q
Queue.

MQOT_REMOTE_Q
Remote queue.

MQOT_TOPIC
Topic.

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

Reason (MQLONG)

This is the reason code resulting from a previous asynchronous put operation on the object specified in *ObjectName*.

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

ResolvedObjectName (MQCHAR48)

This is the name of the object named in *ObjectName* after the local queue manager resolves the name. The name returned is the name of a queue that exists on the queue manager identified by *ResolvedQMgrName*.

This is always an output field. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ResolvedQMgrName (MQCHAR48)

This is the name of the destination queue manager after the local queue manager resolves the name. The name returned is the name of the queue manager that owns the queue identified by *ResolvedObjectName*. *ResolvedQMgrName* can be the name of the local queue manager.

This is always an output field. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

StrucId (MQCHAR4)

This is the structure identifier. The value must be:

MQSTS_STRUC_ID

Identifier for status reporting structure.

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

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

PutSuccessCount (MQLONG)

This is the a count of the number of asynchronous put operations that completed with a completion code of MQCC_OK.

This is always an output field. The initial value of this field is 0.

Version (MQLONG)

This is the structure version number. The value must be:

MQSTS_VERSION_1

Version 1 status reporting structure.

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

MQSTS_CURRENT_VERSION

Current version of status reporting structure.

This is always an input field. The initial value of this field is MQSTS_VERSION_1.

PutWarningCount (MQLONG)

This is a count of the number of asynchronous put operations that completed with a completion code of MQCC_WARNING.

This is always an output field. The initial value of this field is 0.

Initial values and language declarations for MQSTS

Table 76. Initial values of fields in MQSTS

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQSTS_STRUC_ID	'STsb'
<i>Version</i>	MQSTS_CURRENT_VERSION	MQSTS_VERSION_1
<i>CompCode</i>	MQCC_OK	0
<i>Reason</i>	MQRC_NONE	0
<i>PutSuccessCount</i>	None	0
<i>PutWarningCount</i>	None	0
<i>PutFailureCount</i>	None	0
<i>ObjectType</i>	MQOT_Q	1
<i>ObjectName</i>	None	Null string or blanks
<i>ObjectQMgrName</i>	None	Null string or blanks

Table 76. Initial values of fields in MQSTS (continued)

Field name	Name of constant	Value of constant
<i>ResolvedObjectName</i>	None	Null string or blanks
<i>ResolvedQMgrName</i>	None	Null string or blanks

Notes:

1. The symbol `b` represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable `MQSTS_DEFAULT` contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQSTS MySTS = {MQSTS_DEFAULT};
```

C declaration

```
typedef struct tagMQSTS MQSTS;
struct tagMQSTS {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQLONG    CompCode;         /* Completion Code of first error */
    MQLONG    Reason;           /* Reason Code of first error */
    MQLONG    PutSuccessCount;   /* Number of Async calls succeeded */
    MQLONG    PutWarningCount;  /* Number of Async calls had warnings */
    MQLONG    PutFailureCount;  /* Number of Async calls had failures */
    MQLONG    ObjectType;       /* Failing object type */
    MQCHAR48  ObjectName;       /* Failing object name */
    MQCHAR48  ObjectQMgrName;    /* Failing object queue manager name */
    MQCHAR48  ResolvedObjectName; /* Resolved name of destination queue */
    MQCHAR48  ResolvedQMgrName; /* Resolved name of destination qmgr */
};
```

COBOL declaration

```
** MQSTS structure
   10 MQSTS.
      ** Structure identifier
      15 MQSTS-STRUCID          PIC X(4).
      ** Structure version number
      15 MQSTS-VERSION         PIC S9(9) BINARY.
      ** Completion Code
      15 MQSTS-COMPCODE        PIC S9(9) BINARY.
      ** Reason Code
      15 MQSTS-REASON          PIC S9(9) BINARY.
      ** Put success count
      15 MQSTS-PUTSUCCESSCOUNT PIC S9(9) BINARY.
      ** Put Warning count
      15 MQSTS-PUTWARNINGCOUNT PIC S9(9) BINARY.
      ** Put Failure count
      15 MQSTS-PUTFAILURECOUNT PIC S9(9) BINARY.
      ** Object type
      15 MQSTS-OBJECTTYPE      PIC S9(9) BINARY.
      ** Object name
      15 MQSTS-OBJECTNAME      PIC X(48).
      ** Object queue manager name
      15 MQSTS-OBJECTQMGRNAME  PIC X(48).
      ** Resolved object name
      15 MQSTS-RESOLVEDOBJECTNAME PIC X(48).
      ** Resolved object queue manager name
      15 MQSTS-RESOLVEDQMGRNAME PIC X(48).
```

PL/I declaration (z/OS only)

```
dc1
  1 MQSTS based,
  3 StrucId          char(4),          /* Structure identifier */
  3 Version          fixed bin(31), /* Structure version number */
  3 CompCode         fixed bin(31), /* Completion code */
  3 Reason           fixed bin(31), /* Reason code */
  3 PutSuccessCount  fixed bin(31), /* Put success count */
  3 PutWarningCount  fixed bin(31), /* Put warning count */
  3 PutFailureCount  fixed bin(31), /* Put failure count */
  3 ObjectType       fixed bin(31), /* Object type */
  3 ObjectName       char(48), /* Object name */
  3 ObjectQmgrName   char(48), /* Object queue manager */
  3 ResolvedObjectName char(48), /* Resolved Object name */
  3 ResolvedQmgrName char(48); /* Resolved Object queue manager */
```

System/390 assembler declaration (z/OS only)

```
MQSTS          DSECT
MQSTS_STRUCID  DS    CL4  Structure identifier
MQSTS_VERSION  DS    F    Structure version number
MQSTS_COMPCODE DS    F    Completion code
MQSTS_REASON   DS    F    Reason code
MQSTS_PUTSUCCESSCOUNT DS  F  Success count
MQSTS_PUTWARNINGCOUNT DS  F  Warning count
MQSTS_PUTFAILURECOUNT DS  F  Failure count
MQSTS_OBJTYPE  DS    F    Object type
MQSTS_OBJNAME  DS    CL48 Object name
MQSTS_OBJQMGR  DS    CL48 Object queue manager
MQSTS_ROBJNAME DS    CL48 Resolved object name
MQSTS_ROBJQMGR DS    CL48 Resolved object queue manager
```

MQTM – Trigger message

The following table summarizes the fields in the structure.

Table 77. Fields in MQTM

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>QName</i>	Name of triggered queue	QName
<i>ProcessName</i>	Name of process object	ProcessName
<i>TriggerData</i>	Trigger data	TriggerData
<i>ApplType</i>	Application type	ApplType
<i>ApplId</i>	Application identifier	ApplId
<i>EnvData</i>	Environment data	EnvData
<i>UserData</i>	User data	UserData

Overview for MQTM

Purpose: The MQTM structure describes the data in the trigger message that is sent by the queue manager to a trigger-monitor application when a trigger event occurs for a queue.

This structure is part of the WebSphere MQ Trigger Monitor Interface (TMI), which is one of the WebSphere MQ framework interfaces.

Format name: MQFMT_TRIGGER.

Character set and encoding: Character data in MQTM is in the character set of the queue manager that generates the MQTM. Numeric data in MQTM is in the machine encoding of the queue manager that generates the MQTM.

The character set and encoding of the MQTM are given by the *CodedCharSetId* and *Encoding* fields in:

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

Usage: A trigger-monitor application might need to pass some or all of the information in the trigger message to the application that the trigger-monitor application starts. Information that might be needed by the started application includes *QName*, *TriggerData*, and *UserData*. The trigger-monitor application can pass the MQTM structure directly to the started application, or pass an MQTMC2 structure instead, depending on what is permitted by the environment and convenient for the started application. For information about MQTMC2, see “MQTMC2 – Trigger message 2 (character format)” on page 373.

- On z/OS, for an MQAT_CICS application that is started using the CKTI transaction, the entire trigger message structure MQTM is made available to the started transaction; the information can be retrieved by using the EXEC CICS RETRIEVE command.
- On i5/OS, the trigger-monitor application provided with WebSphere MQ passes an MQTMC2 structure to the started application.

For information about using triggers, see the *WebSphere MQ Application Programming Guide*.

MQMD for a trigger message: The fields in the MQMD of a trigger message generated by the queue manager are set as follows:

Field in MQMD	Value used
<i>StrucId</i>	MQMD_STRUC_ID
<i>Version</i>	MQMD_VERSION_1
<i>Report</i>	MQRO_NONE
<i>MsgType</i>	MQMT_DATAGRAM
<i>Expiry</i>	MQEI_UNLIMITED
<i>Feedback</i>	MQFB_NONE
<i>Encoding</i>	MQENC_NATIVE
<i>CodedCharSetId</i>	Queue manager's <i>CodedCharSetId</i> attribute
<i>Format</i>	MQFMT_TRIGGER
<i>Priority</i>	Initiation queue's <i>DefPriority</i> attribute
<i>Persistence</i>	MQPER_NOT_PERSISTENT
<i>MsgId</i>	A unique value
<i>CorrelId</i>	MQCI_NONE
<i>BackoutCount</i>	0
<i>ReplyToQ</i>	Blanks
<i>ReplyToQMgr</i>	Name of queue manager
<i>UserIdentifier</i>	Blanks
<i>AccountingToken</i>	MQACT_NONE
<i>ApplIdentityData</i>	Blanks
<i>PutApplType</i>	MQAT_QMGR, or as appropriate for the message channel agent
<i>PutApplName</i>	First 28 bytes of the queue-manager name
<i>PutDate</i>	Date when trigger message is sent
<i>PutTime</i>	Time when trigger message is sent

Field in MQMD	Value used
<i>ApplOriginData</i>	Blanks

An application that generates a trigger message is recommended to set similar values, except for the following:

- The *Priority* field can be set to MQPRI_PRIORITY_AS_Q_DEF (the queue manager will change this to the default priority for the initiation queue when the message is put).
- The *ReplyToQMGr* field can be set to blanks (the queue manager will change this to the name of the local queue manager when the message is put).
- Set the context fields as appropriate for the application.

Fields for MQTM

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

ApplId (MQCHAR256)

This is a character string that identifies the application to be started, and is used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the *ApplId* attribute of the process object identified by the *ProcessName* field; see “Attributes for process definitions” on page 654 for details of this attribute. The content of this data is of no significance to the queue manager.

The meaning of *ApplId* is determined by the trigger-monitor application. The trigger monitor provided by WebSphere MQ requires *ApplId* to be the name of an executable program. The following notes apply to the environments indicated:

- On z/OS, *ApplId* is:
 - A CICS transaction identifier, for applications started using the CICS trigger-monitor transaction CKTI
 - An IMS transaction identifier, for applications started using the IMS trigger monitor CSQQTRMN
- On Windows systems, the program name can be prefixed with a drive and directory path.
- On i5/OS, the program name can be prefixed with a library name and / character.
- On UNIX systems, the program name can be prefixed with a directory path.

The length of this field is given by MQ_PROCESS_APPL_ID_LENGTH. The initial value of this field is the null string in C, and 256 blank characters in other programming languages.

ApplType (MQLONG)

This identifies the nature of the program to be started, and is used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the *ApplType* attribute of the process object identified by the *ProcessName* field; see “Attributes for process definitions” on page 654 for details of this attribute. The content of this data is of no significance to the queue manager.

ApplType can have one of the following standard values. User-defined types can also be used, but should be restricted to values in the range MQAT_USER_FIRST through MQAT_USER_LAST:

MQAT_AIX
AIX application (same value as MQAT_UNIX).

|
| **MQAT_BATCH**
Batch application

|
| **MQAT_BROKER**
Broker application

MQAT_CICS
CICS transaction.

|
| **MQAT_CICS_BRIDGE**
CICS bridge application.

MQAT_CICS_VSE
CICS/VSE transaction.

MQAT_DOS
WebSphere MQ client application on PC DOS.

MQAT_IMS
IMS application.

|
| **MQAT_IMS_BRIDGE**
IMS bridge application.

|
| **MQAT_JAVA**
Java application.

MQAT_MVS
MVS or TSO application (same value as MQAT_ZOS).

MQAT_NOTES_AGENT
Lotus Notes Agent application.

MQAT_NSK
Compaq NonStop Kernel application.

MQAT_OS2
OS/2 or Presentation Manager application.

MQAT_OS390
OS/390 application (same value as MQAT_ZOS).

MQAT_OS400
i5/OS application.

|
| **MQAT_RRS_BATCH**
RRS batch application.

MQAT_UNIX
UNIX application.

|
| **MQAT_UNKNOWN**
Application of unknown type.

|
| **MQAT_USER**
User-defined application type.

MQAT_VMS
Digital OpenVMS application.

MQAT_VOS
Stratus VOS application.

MQAT_WINDOWS
16-bit Windows application.

MQAT_WINDOWS_NT
32-bit Windows application.

MQAT_WLM
z/OS workload manager application.

MQAT_XCF
XCF.

MQAT_ZOS
z/OS application.

MQAT_USER_FIRST
Lowest value for user-defined application type.

MQAT_USER_LAST
Highest value for user-defined application type.

The initial value of this field is 0.

EnvData (MQCHAR128)

This is a character string that contains environment-related information pertaining to the application to be started, and is used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the *EnvData* attribute of the process object identified by the *ProcessName* field; see “Attributes for process definitions” on page 654 for details of this attribute. The content of this data is of no significance to the queue manager.

On z/OS, for a CICS application started using the CKTI transaction, or an IMS application to be started using the CSQQTRMN transaction, this information is not used.

The length of this field is given by MQ_PROCESS_ENV_DATA_LENGTH. The initial value of this field is the null string in C, and 128 blank characters in other programming languages.

ProcessName (MQCHAR48)

This is the name of the queue-manager process object specified for the triggered queue, and can be used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the *ProcessName* attribute of the queue identified by the *QName* field; see “Attributes for queues” on page 615 for details of this attribute.

Names that are shorter than the defined length of the field are always padded to the right with blanks; they are not ended prematurely by a null character.

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

QName (MQCHAR48)

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 “Attributes for queues” on page 615 for details of this attribute.

Names that are shorter than the defined length of the field are padded to the right with blanks; they are not ended prematurely by a null character.

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

StruId (MQCHAR4)

This is the structure identifier. The value must be:

MQTM_STRUC_ID

Identifier for trigger message structure.

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

The initial value of this field is MQTM_STRUC_ID.

TriggerData (MQCHAR64)

This is free-format data for use by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the *TriggerData* attribute of the queue identified by the *QName* field; see “Attributes for queues” on page 615 for details of this attribute. The content of this data is of no significance to the queue manager.

On z/OS, for a CICS application started using the CKTI transaction, this information is not used.

The length of this field is given by MQ_TRIGGER_DATA_LENGTH. The initial value of this field is the null string in C, and 64 blank characters in other programming languages.

UserData (MQCHAR128)

This is a character string that contains user information relevant to the application to be started, and is used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the *UserData* attribute of the process object identified by the *ProcessName* field; see “Attributes for process definitions” on page 654 for details of this attribute. The content of this data is of no significance to the queue manager.

The length of this field is given by MQ_PROCESS_USER_DATA_LENGTH. The initial value of this field is the null string in C, and 128 blank characters in other programming languages.

Version (MQLONG)

This is the structure version number. The value must be:

MQTM_VERSION_1

Version number for trigger message structure.

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

MQTM_CURRENT_VERSION

Current version of trigger message structure.

The initial value of this field is MQTM_VERSION_1.

Initial values and language declarations for MQTM

Table 78. Initial values of fields in MQTM for MQTM

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQTM_STRUC_ID	' TMbb '
<i>Version</i>	MQTM_VERSION_1	1
<i>QName</i>	None	Null string or blanks
<i>ProcessName</i>	None	Null string or blanks
<i>TriggerData</i>	None	Null string or blanks
<i>ApplType</i>	None	0
<i>ApplId</i>	None	Null string or blanks
<i>EnvData</i>	None	Null string or blanks
<i>UserData</i>	None	Null string or blanks

Notes:

1. The symbol **b** represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQTM_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

```
MQTM MyTM = {MQTM_DEFAULT};
```

C declaration

```
typedef struct tagMQTM MQTM;  
struct tagMQTM {  
    MQCHAR4   StrucId;    /* Structure identifier */  
    MQLONG    Version;    /* Structure version number */  
    MQCHAR48   QName;     /* Name of triggered queue */  
    MQCHAR48   ProcessName; /* Name of process object */  
    MQCHAR64   TriggerData; /* Trigger data */  
    MQLONG     ApplType;   /* Application type */  
    MQCHAR256  ApplId;     /* Application identifier */  
    MQCHAR128  EnvData;    /* Environment data */  
    MQCHAR128  UserData;   /* User data */  
};
```

COBOL declaration

```
** MQTM structure  
10 MQTM.  
** Structure identifier  
15 MQTM-STRUCID PIC X(4).  
** Structure version number
```

```

15 MQTM-VERSION    PIC S9(9) BINARY.
**   Name of triggered queue
15 MQTM-QNAME      PIC X(48).
**   Name of process object
15 MQTM-PROCESSNAME PIC X(48).
**   Trigger data
15 MQTM-TRIGGERDATA PIC X(64).
**   Application type
15 MQTM-APPLTYPE   PIC S9(9) BINARY.
**   Application identifier
15 MQTM-APPLID     PIC X(256).
**   Environment data
15 MQTM-ENVDATA    PIC X(128).
**   User data
15 MQTM-USERDATA   PIC X(128).

```

PL/I declaration

```

dcl
1 MQTM based,
3 StrucId      char(4),          /* Structure identifier */
3 Version      fixed bin(31),   /* Structure version number */
3 QName        char(48),        /* Name of triggered queue */
3 ProcessName  char(48),        /* Name of process object */
3 TriggerData  char(64),        /* Trigger data */
3 ApplType     fixed bin(31),   /* Application type */
3 ApplId       char(256),       /* Application identifier */
3 EnvData      char(128),       /* Environment data */
3 UserData     char(128);       /* User data */

```

System/390 assembler declaration

```

MQTM          DSECT
MQTM_STRUCID  DS   CL4   Structure identifier
MQTM_VERSION  DS   F     Structure version number
MQTM_QNAME    DS   CL48  Name of triggered queue
MQTM_PROCESSNAME DS CL48  Name of process object
MQTM_TRIGGERDATA DS CL64  Trigger data
MQTM_APPLTYPE DS   F     Application type
MQTM_APPLID   DS   CL256 Application identifier
MQTM_ENVDATA  DS   CL128 Environment data
MQTM_USERDATA DS   CL128 User data
*
MQTM_LENGTH   EQU  *-MQTM
              ORG  MQTM
MQTM_AREA     DS   CL(MQTM_LENGTH)

```

Visual Basic declaration

```

Type MQTM
  StrucId As String*4   'Structure identifier'
  Version As Long       'Structure version number'
  QName As String*48   'Name of triggered queue'
  ProcessName As String*48 'Name of process object'
  TriggerData As String*64 'Trigger data'
  ApplType As Long     'Application type'
  ApplId As String*256 'Application identifier'
  EnvData As String*128 'Environment data'
  UserData As String*128 'User data'
End Type

```

MQTM C2 – Trigger message 2 (character format)

The following table summarizes the fields in the structure.

Table 79. Fields in MQTMC2

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>QName</i>	Name of triggered queue	QName
<i>ProcessName</i>	Name of process object	ProcessName
<i>TriggerData</i>	Trigger data	TriggerData
<i>ApplType</i>	Application type	ApplType
<i>ApplId</i>	Application identifier	ApplId
<i>EnvData</i>	Environment data	EnvData
<i>UserData</i>	User data	UserData
<i>QMgrName</i>	Queue manager name	QMgrName

Overview for MQTMC2

Purpose: When a trigger-monitor application retrieves a trigger message (MQTM) from an initiation queue, the trigger monitor might need to pass some or all of the information in the trigger message to the application that the trigger monitor starts.

Information that the started application might need includes *QName*, *TriggerData*, and *UserData*. The trigger monitor application can pass the MQTM structure directly to the started application, or pass an MQTMC2 structure instead, depending on what is permitted by the environment and convenient for the started application.

This structure is part of the WebSphere MQ Trigger Monitor Interface (TMI), which is one of the WebSphere MQ framework interfaces.

Character set and encoding: Character data in MQTMC2 is in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute.

Usage: The MQTMC2 structure is very similar to the format of the MQTM structure. The difference is that the non-character fields in MQTM are changed in MQTMC2 to character fields of the same length, and the queue manager name is added at the end of the structure.

- On z/OS, for an MQAT_IMS application that is started using the CSQQTRMN application, an MQTMC2 structure is made available to the started application.
- On i5/OS, the trigger monitor application provided with WebSphere MQ passes an MQTMC2 structure to the started application.

Fields for MQTMC2

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

ApplId (MQCHAR256)

Application identifier.

See the *ApplId* field in the MQTM structure.

AppType (MQCHAR4)

Application type.

This field always contains blanks, whatever the value in the *AppType* field in the MQTM structure of the original trigger message.

EnvData (MQCHAR128)

Environment data.

See the *EnvData* field in the MQTM structure.

ProcessName (MQCHAR48)

Name of process object.

See the *ProcessName* field in the MQTM structure.

QMgrName (MQCHAR48)

Queue manager name.

This is the name of the queue manager at which the trigger event occurred.

QName (MQCHAR48)

Name of triggered queue.

See the *QName* field in the MQTM structure.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQTMC_STRUC_ID

Identifier for trigger message (character format) structure.

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

TriggerData (MQCHAR64)

Trigger data.

See the *TriggerData* field in the MQTM structure.

UserData (MQCHAR128)

User data.

See the *UserData* field in the MQTM structure.

Version (MQCHAR4)

Structure version number.

The value must be:

MQTMC_VERSION_2

Version 2 trigger message (character format) structure.

For the C programming language, the constant MQTMC_VERSION_2_ARRAY is also defined; this has the same value as MQTMC_VERSION_2, but is an array of characters instead of a string.

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

MQTMC_CURRENT_VERSION

Current version of trigger message (character format) structure.

Initial values and language declarations for MQTMC2

Table 80. Initial values of fields in MQTMC2 for MQTMC2

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQTMC_STRUC_ID	'TMCb'
<i>Version</i>	MQTMC_VERSION_2	'bbb2'
<i>QName</i>	None	Null string or blanks
<i>ProcessName</i>	None	Null string or blanks
<i>TriggerData</i>	None	Null string or blanks
<i>ApplType</i>	None	Blanks
<i>ApplId</i>	None	Null string or blanks
<i>EnvData</i>	None	Null string or blanks
<i>UserData</i>	None	Null string or blanks
<i>QMgrName</i>	None	Null string or blanks

Notes:

1. The symbol **b** represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQTMC2_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

```
MQTMC2 MyTMC = {MQTMC2_DEFAULT};
```

C declaration

```
typedef struct tagMQTMC2 MQTMC2;
struct tagMQTMC2 {
    MQCHAR4    StrucId;        /* Structure identifier */
    MQCHAR4    Version;       /* Structure version number */
    MQCHAR48   QName;         /* Name of triggered queue */
    MQCHAR48   ProcessName;   /* Name of process object */
    MQCHAR64   TriggerData;   /* Trigger data */
    MQCHAR4    ApplType;      /* Application type */
    MQCHAR256  ApplId;        /* Application identifier */
    MQCHAR128  EnvData;       /* Environment data */
    MQCHAR128  UserData;      /* User data */
    MQCHAR48   QMgrName;      /* Queue manager name */
};
```

COBOL declaration

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

PL/I declaration

```
dc1
1 MQTMC2 based,
3 StrucId      char(4), /* Structure identifier */
3 Version      char(4), /* Structure version number */
3 QName        char(48), /* Name of triggered queue */
3 ProcessName  char(48), /* Name of process object */
3 TriggerData  char(64), /* Trigger data */
3 ApplType     char(4), /* Application type */
3 ApplId       char(256), /* Application identifier */
3 EnvData      char(128), /* Environment data */
3 UserData     char(128), /* User data */
3 QMgrName     char(48); /* Queue manager name */
```

System/390 assembler declaration

```
MQTMCT          DSECT
MQTMCT_STRUCID  DS   CL4   Structure identifier
MQTMCT_VERSION DS   CL4   Structure version number
MQTMCT_QNAME    DS   CL48  Name of triggered queue
MQTMCT_PROCESSNAME DS CL48  Name of process object
MQTMCT_TRIGGERDATA DS CL64  Trigger data
MQTMCT_APPLTYPE DS   CL4   Application type
MQTMCT_APPLID   DS   CL256 Application identifier
MQTMCT_ENVDATA  DS   CL128 Environment data
MQTMCT_USERDATA DS   CL128 User data
MQTMCT_QMGRNAME DS   CL48  Queue manager name
*
MQTMCT_LENGTH  EQU  *-MQTMCT
                ORG  MQTMCT
MQTMCT_AREA    DS   CL(MQTMCT_LENGTH)
```

Visual Basic declaration

```
Type MQTMC2
    StrucId As String*4 'Structure identifier'
    Version As String*4 'Structure version number'
    QName As String*48 'Name of triggered queue'
    ProcessName As String*48 'Name of process object'
    TriggerData As String*64 'Trigger data'
    ApplType As String*4 'Application type'
    ApplId As String*256 'Application identifier'
```

```

EnvData      As String*128 'Environment data'
UserData     As String*128 'User data'
QMgrName    As String*48  'Queue manager name'
End Type

```

MQWIH – Work information header

The following table summarizes the fields in the structure.

Table 81. Fields in MQWIH

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>StrucLength</i>	Length of MQWIH structure	StrucLength
<i>Encoding</i>	Numeric encoding of data that follows MQWIH	Encoding
<i>CodedCharSetId</i>	Character-set identifier of data that follows MQWIH	CodedCharSetId
<i>Format</i>	Format name of data that follows MQWIH	Format
<i>Flags</i>	Flags	Flags
<i>ServiceName</i>	Service name	ServiceName
<i>ServiceStep</i>	Service step name	ServiceStep
<i>MsgToken</i>	Message token	MsgToken
<i>Reserved</i>	Reserved	Reserved

Overview for MQWIH

Availability: All WebSphere MQ systems, plus WebSphere MQ clients connected to these systems.

Purpose: The MQWIH structure describes the information that must be present at the start of a message that is to be handled by the z/OS workload manager.

Format name: MQFMT_WORK_INFO_HEADER.

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

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

Usage: If a message is to be processed by the z/OS workload manager, the message must begin with an MQWIH structure.

Fields for MQWIH

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

CodedCharSetId (MQLONG)

This specifies the character set identifier of the data that follows the MQWIH structure; it does not apply to character data in the MQWIH structure itself.

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

MQCCSI_INHERIT

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

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

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

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG)

This specifies the numeric encoding of the data that follows the MQWIH structure; it does not apply to numeric data in the MQWIH structure itself.

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

The initial value of this field is 0.

Flags (MQLONG)

The value must be:

MQWIH_NONE

No flags.

The initial value of this field is MQWIH_NONE.

Format (MQCHAR8)

This specifies the format name of the data that follows the MQWIH structure.

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

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

MsgToken (MQBYTE16)

This is a message token that uniquely identifies the message.

For the MQPUT and MQPUT1 calls, this field is ignored. The length of this field is given by MQ_MSG_TOKEN_LENGTH. The initial value of this field is MQMTOK_NONE.

Reserved (MQCHAR32)

This is a reserved field; it must be blank.

ServiceName (MQCHAR32)

This is the name of the service that is to process the message.

The length of this field is given by MQ_SERVICE_NAME_LENGTH. The initial value of this field is 32 blank characters.

ServiceStep (MQCHAR8)

This is the name of the step of *ServiceName* to which the message relates.

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

StrucId (MQCHAR4)

This is the structure identifier. The value must be:

MQWIH_STRUC_ID

Identifier for work information header structure.

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

The initial value of this field is MQWIH_STRUC_ID.

StrucLength (MQLONG)

This is the length of the MQWIH structure. The value must be:

MQWIH_LENGTH_1

Length of version-1 work information header structure.

The following constant specifies the length of the current version:

MQWIH_CURRENT_LENGTH

Length of current version of work information header structure.

The initial value of this field is MQWIH_LENGTH_1.

Version (MQLONG)

This is the structure version number. The value must be:

MQWIH_VERSION_1

Version-1 work information header structure.

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

MQWIH_CURRENT_VERSION

Current version of work information header structure.

The initial value of this field is MQWIH_VERSION_1.

Initial values and language declarations for MQWIH

Table 82. Initial values of fields in MQWIH for MQWIH

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

Notes:

1. The symbol *b* represents a single blank character.
2. In the C programming language, the macro variable `MQWIH_DEFAULT` contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

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

C declaration

```
typedef struct tagMQWIH MQWIH;
struct tagMQWIH {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQLONG    StrucLength;      /* Length of MQWIH structure */
    MQLONG    Encoding;         /* Numeric encoding of data that follows
                               MQWIH */
    MQLONG    CodedCharSetId;   /* Character-set identifier of data that
                               follows MQWIH */
    MQCHAR8   Format;           /* Format name of data that follows
                               MQWIH */
    MQLONG    Flags;           /* Flags */
    MQCHAR32  ServiceName;     /* Service name */
    MQCHAR8   ServiceStep;     /* Service step name */
    MQBYTE16  MsgToken;        /* Message token */
    MQCHAR32  Reserved;        /* Reserved */
};
```

COBOL declaration

```
** MQWIH structure
10 MQWIH.
** Structure identifier
15 MQWIH-STRUCID PIC X(4).
** Structure version number
15 MQWIH-VERSION PIC S9(9) BINARY.
** Length of MQWIH structure
15 MQWIH-STRUCLength PIC S9(9) BINARY.
** Numeric encoding of data that follows MQWIH
15 MQWIH-ENCODING PIC S9(9) BINARY.
** Character-set identifier of data that follows MQWIH
```

```

    15 MQWIH-CODEDCHARSETID PIC S9(9) BINARY.
**   Format name of data that follows MQWIH
    15 MQWIH-FORMAT        PIC X(8).
**   Flags
    15 MQWIH-FLAGS        PIC S9(9) BINARY.
**   Service name
    15 MQWIH-SERVICENAME  PIC X(32).
**   Service step name
    15 MQWIH-SERVICESTEP PIC X(8).
**   Message token
    15 MQWIH-MSGTOKEN    PIC X(16).
**   Reserved
    15 MQWIH-RESERVED    PIC X(32).

```

PL/I declaration

```

dcl
1 MQWIH based,
  3 StrucId      char(4),      /* Structure identifier */
  3 Version      fixed bin(31), /* Structure version number */
  3 StrucLength  fixed bin(31), /* Length of MQWIH structure */
  3 Encoding     fixed bin(31), /* Numeric encoding of data that
                                follows MQWIH */
  3 CodedCharSetId fixed bin(31), /* Character-set identifier of data
                                that follows MQWIH */
  3 Format        char(8),      /* Format name of data that follows
                                MQWIH */
  3 Flags        fixed bin(31), /* Flags */
  3 ServiceName  char(32),     /* Service name */
  3 ServiceStep  char(8),      /* Service step name */
  3 MsgToken     char(16),     /* Message token */
  3 Reserved     char(32);     /* Reserved */

```

System/390 assembler declaration

```

MQWIH          DSECT
MQWIH_STRUCID  DS   CL4   Structure identifier
MQWIH_VERSION  DS   F     Structure version number
MQWIH_STRUCLNGTH DS   F     Length of MQWIH structure
MQWIH_ENCODING DS   F     Numeric encoding of data that follows
*               MQWIH
MQWIH_CODEDCHARSETID DS   F     Character-set identifier of data that
*               follows MQWIH
MQWIH_FORMAT   DS   CL8   Format name of data that follows MQWIH
MQWIH_FLAGS    DS   F     Flags
MQWIH_SERVICENAME DS   CL32 Service name
MQWIH_SERVICESTEP DS   CL8   Service step name
MQWIH_MSGTOKEN DS   XL16  Message token
MQWIH_RESERVED DS   CL32  Reserved
*
MQWIH_LENGTH   EQU   *-MQWIH
               ORG   MQWIH
MQWIH_AREA     DS   CL(MQWIH_LENGTH)

```

Visual Basic declaration

```

Type MQWIH
  StrucId      As String*4  'Structure identifier'
  Version      As Long      'Structure version number'
  StrucLength  As Long      'Length of MQWIH structure'
  Encoding     As Long      'Numeric encoding of data that follows'
                                'MQWIH'
  CodedCharSetId As Long    'Character-set identifier of data that'
                                'follows MQWIH'
  Format        As String*8  'Format name of data that follows MQWIH'
  Flags        As Long      'Flags'
  ServiceName  As String*32 'Service name'

```

```

ServiceStep  As String*8  'Service step name'
MsgToken    As MQBYTE16  'Message token'
Reserved    As String*32  'Reserved'
End Type

```

MQXP – Exit parameter block

The following table summarizes the fields in the structure.

Table 83. Fields in MQXP

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>ExitId</i>	Exit identifier	ExitId
<i>ExitReason</i>	Reason for invocation of exit	ExitReason
<i>ExitResponse</i>	Response from exit	ExitResponse
<i>ExitCommand</i>	API call code	ExitCommand
<i>ExitParmCount</i>	Parameter count	ExitParmCount
<i>ExitUserArea</i>	User area	ExitUserArea

Overview for MQXP

Availability: z/OS.

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

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

Fields for MQXP

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

ExitCommand (MQLONG)

This field is set on entry to the exit routine. It identifies the API call that caused the exit to be invoked:

MQXC_MQBACK
The MQBACK call.

MQXC_MQCLOSE
The MQCLOSE call.

MQXC_MQCMIT
The MQCMIT call.

MQXC_MQGET
The MQGET call.

MQXC_MQINQ
The MQINQ call.

MQXC_MQOPEN
The MQOPEN call.

MQXC_MQPUT
The MQPUT call.

MQXC_MQPUT1
The MQPUT1 call.

MQXC_MQSET
The MQSET call.

This is an input field to the exit.

ExitId (MQLONG)

This is set on entry to the exit routine, and indicates the type of exit:

MQXT_API_CROSSING_EXIT
API-crossing exit for CICS.

This is an input field to the exit.

ExitParmCount (MQLONG)

This field is set on entry to the exit routine. It contains the number of parameters that the MQ call takes. These are:

Call name	Number of parameters
MQBACK	3
MQCLOSE	5
MQCMIT	3
MQGET	9
MQINQ	10
MQOPEN	6
MQPUT	8
MQPUT1	8
MQSET	10

This is an input field to the exit.

ExitReason (MQLONG)

This is set on entry to the exit routine. For the API-crossing exit it indicates whether the routine is called before or after execution of the API call:

MQXR_BEFORE
Before API execution.

MQXR_AFTER
After API execution.

This is an input field to the exit.

ExitResponse (MQLONG)

The value is set by the exit to communicate with the caller. The following values are defined:

MQXCC_OK

Exit completed successfully.

MQXCC_SUPPRESS_FUNCTION

Suppress function.

When this value is set by an API-crossing exit called *before* the API call, the API call is not performed. The *CompCode* for the call is set to MQCC_FAILED, the *Reason* is set to MQRC_SUPPRESSED_BY_EXIT, and all other parameters remain as the exit left them.

When this value is set by an API-crossing exit called *after* the API call, it is ignored by the queue manager.

MQXCC_SKIP_FUNCTION

Skip function.

When this value is set by an API-crossing exit called *before* the API call, the API call is not performed; the *CompCode* and *Reason* and all other parameters remain as the exit left them.

When this value is set by an API-crossing exit called *after* the API call, it is ignored by the queue manager.

This is an output field from the exit.

ExitUserArea (MQBYTE16)

This is a field that is available for the exit to use. It is initialized to binary zero for the length of the field before the first invocation of the exit for the task, and thereafter any changes made to this field by the exit are preserved across invocations of the exit. The following value is defined:

MQXUA_NONE

No user information.

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

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

The length of this field is given by MQ_EXIT_USER_AREA_LENGTH. This is an input/output field to the exit.

Reserved (MQLONG)

This is a reserved field. Its value is not significant to the exit.

StrucId (MQCHAR4)

This is the structure identifier. The value must be:

MQXP_STRUC_ID

Identifier for exit parameter structure.

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

This is an input field to the exit.

Version (MQLONG)

This is the structure version number. The value must be:

`MQXP_VERSION_1`

Version number for exit parameter-block structure.

Note: When a new version of this structure is introduced, the layout of the existing part is not changed. The exit must therefore check that the version number is equal to or greater than the lowest version that contains the fields that the exit needs to use.

This is an input field to the exit.

Language declarations

This structure is supported in the following programming languages.

C declaration

```
typedef struct tagMQXP MQXP;
struct tagMQXP {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQLONG    ExitId;           /* Exit identifier */
    MQLONG    ExitReason;       /* Reason for invocation of exit */
    MQLONG    ExitResponse;     /* Response from exit */
    MQLONG    ExitCommand;      /* API call code */
    MQLONG    ExitParmCount;    /* Parameter count */
    MQLONG    Reserved;        /* Reserved */
    MQBYTE16  ExitUserArea;     /* User area */
};
```

COBOL declaration

```
** MQXP structure
  10 MQXP.
**   Structure identifier
  15 MQXP-STRUCID    PIC X(4).
**   Structure version number
  15 MQXP-VERSION   PIC S9(9) BINARY.
**   Exit identifier
  15 MQXP-EXITID    PIC S9(9) BINARY.
**   Reason for invocation of exit
  15 MQXP-EXITREASON PIC S9(9) BINARY.
**   Response from exit
  15 MQXP-EXITRESPONSE PIC S9(9) BINARY.
**   API call code
  15 MQXP-EXITCOMMAND PIC S9(9) BINARY.
**   Parameter count
  15 MQXP-EXITPARMCOUNT PIC S9(9) BINARY.
**   Reserved
  15 MQXP-RESERVED   PIC S9(9) BINARY.
**   User area
  15 MQXP-EXITUSERAREA PIC X(16).
```

PL/I declaration

```
dc1
1 MQXP based,
3 StrucId      char(4),      /* Structure identifier */
3 Version      fixed bin(31), /* Structure version number */
3 ExitId       fixed bin(31), /* Exit identifier */
3 ExitReason   fixed bin(31), /* Reason for invocation of exit */
3 ExitResponse fixed bin(31), /* Response from exit */
3 ExitCommand  fixed bin(31), /* API call code */
3 ExitParmCount fixed bin(31), /* Parameter count */
3 Reserved     fixed bin(31), /* Reserved */
3 ExitUserArea char(16);    /* User area */
```

System/390 assembler declaration

```
MQXP          DSECT
MQXP_STRUCID  DS   CL4   Structure identifier
MQXP_VERSION  DS   F     Structure version number
MQXP_EXITID   DS   F     Exit identifier
MQXP_EXITREASON DS   F   Reason for invocation of exit
MQXP_EXITRESPONSE DS   F Response from exit
MQXP_EXITCOMMAND DS   F  API call code
MQXP_EXITPARMCOUNT DS   F Parameter count
MQXP_RESERVED DS   F     Reserved
MQXP_EXITUSERAREA DS  XL16 User area
*
MQXP_LENGTH   EQU   *-MQXP
              ORG   MQXP
MQXP_AREA     DS   CL(MQXP_LENGTH)
```

MQXQH – Transmission-queue header

The following table summarizes the fields in the structure.

Table 84. Fields in MQXQH

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>RemoteQName</i>	Name of destination queue	RemoteQName
<i>RemoteQMgrName</i>	Name of destination queue manager	RemoteQMgrName
<i>MsgDesc</i>	Original message descriptor	MsgDesc

Overview for MQXQH

Availability: All WebSphere MQ systems and WebSphere MQ clients.

Purpose: The MQXQH structure describes the information that is prefixed to the application message data of messages when they are on transmission queues. A transmission queue is a special type of local queue that temporarily holds messages destined for remote queues (that is, destined for queues that do not belong to the local queue manager). A transmission queue is denoted by the *Usage* queue attribute having the value MQUS_TRANSMISSION.

Format name: MQFMT_XMIT_Q_HEADER.

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

Set the character set and encoding of the MQXQH into the *CodedCharSetId* and *Encoding* fields in:

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

Usage: A message that is on a transmission queue has *two* message descriptors:

- One message descriptor is stored separately from the message data; this is called the *separate message descriptor*, and is generated by the queue manager when the message is placed on the transmission queue. Some of the fields in the separate message descriptor are copied from the message descriptor provided by the application on the MQPUT or MQPUT1 call (see below for details).

The separate message descriptor is the one that is returned to the application in the *MsgDesc* parameter of the MQGET call when the message is removed from the transmission queue.

- A second message descriptor is stored within the MQXQH structure as part of the message data; this is called the *embedded message descriptor*, and is a copy of the message descriptor that was provided by the application on the MQPUT or MQPUT1 call (with minor variations; see below for details).

The embedded message descriptor is always a version-1 MQMD. If the message put by the application has nondefault values for one or more of the version-2 fields in the MQMD, an MQMDE structure follows the MQXQH, and is in turn followed by the application message data (if any). The MQMDE is either:

- Generated by the queue manager (if the application uses a version-2 MQMD to put the message), or
- Already present at the start of the application message data (if the application uses a version-1 MQMD to put the message).

The embedded message descriptor is the one that is returned to the application in the *MsgDesc* parameter of the MQGET call when the message is removed from the final destination queue.

Fields in the separate message descriptor: The fields in the separate message descriptor are set by the queue manager as shown below. If the queue manager does not support the version-2 MQMD, a version-1 MQMD is used without loss of function.

Field in separate MQMD	Value used
<i>StrucId</i>	MQMD_STRUC_ID
<i>Version</i>	MQMD_VERSION_2
<i>Report</i>	Copied from the embedded message descriptor, but with the bits identified by MQRO_ACCEPT_UNSUP_IF_XMIT_MASK set to zero. (This prevents a COA or COD report message being generated when a message is placed on or removed from a transmission queue.)
<i>MsgType</i>	Copied from the embedded message descriptor.
<i>Expiry</i>	Copied from the embedded message descriptor.
<i>Feedback</i>	Copied from the embedded message descriptor.
<i>Encoding</i>	MQENC_NATIVE (see note below)
<i>CodedCharSetId</i>	Queue manager's <i>CodedCharSetId</i> attribute.
<i>Format</i>	MQFMT_XMIT_Q_HEADER
<i>Priority</i>	Copied from the embedded message descriptor.
<i>Persistence</i>	Copied from the embedded message descriptor.

Field in separate MQMD	Value used
<i>MsgId</i>	A new value is generated by the queue manager. This message identifier is different from the <i>MsgId</i> that the queue manager may have generated for the embedded message descriptor (see above).
<i>CorrelId</i>	The <i>MsgId</i> from the embedded message descriptor.
<i>BackoutCount</i>	0
<i>ReplyToQ</i>	Copied from the embedded message descriptor.
<i>ReplyToQMgr</i>	Copied from the embedded message descriptor.
<i>UserIdentifier</i>	Copied from the embedded message descriptor.
<i>AccountingToken</i>	Copied from the embedded message descriptor.
<i>ApplIdentityData</i>	Copied from the embedded message descriptor.
<i>PutApplType</i>	MQAT_QMGR
<i>PutApplName</i>	First 28 bytes of the queue-manager name.
<i>PutDate</i>	Date when message was put on transmission queue.
<i>PutTime</i>	Time when message was put on transmission queue.
<i>ApplOriginData</i>	Blanks
<i>GroupId</i>	MQGI_NONE
<i>MsgSeqNumber</i>	1
<i>Offset</i>	0
<i>MsgFlags</i>	MQMF_NONE
<i>OriginalLength</i>	MQOL_UNDEFINED

- On Windows, the value of MQENC_NATIVE for Micro Focus COBOL differs from the value for C. The value in the *Encoding* field in the separate message descriptor is always the value for C in these environments; this value is 546 in decimal. Also, the integer fields in the MQXQH structure are in the encoding that corresponds to this value (the native Intel® encoding).

Fields in the embedded message descriptor: The fields in the embedded message descriptor have the same values as those in the *MsgDesc* parameter of the MQPUT or MQPUT1 call, with the exception of the following:

- The *Version* field always has the value MQMD_VERSION_1.
- If the *Priority* field has the value MQPRI_PRIORITY_AS_Q_DEF, it is replaced by the value of the queue's *DefPriority* attribute.
- If the *Persistence* field has the value MQPER_PERSISTENCE_AS_Q_DEF, it is replaced by the value of the queue's *DefPersistence* attribute.
- If the *MsgId* field has the value MQMI_NONE, or the MQPMO_NEW_MSG_ID option was specified, or the message is a distribution-list message, *MsgId* is replaced by a new message identifier generated by the queue manager.
When a distribution-list message is split into smaller distribution-list messages placed on different transmission queues, the *MsgId* field in each of the new embedded message descriptors is the same as that in the original distribution-list message.
- If the MQPMO_NEW_CORREL_ID option was specified, *CorrelId* is replaced by a new correlation identifier generated by the queue manager.
- The context fields are set as indicated by the MQPMO_*_CONTEXT options specified in the *PutMsgOpts* parameter; the context fields are:
 - *AccountingToken*
 - *ApplIdentityData*
 - *ApplOriginData*
 - *PutApplName*
 - *PutApplType*

- *PutDate*
- *PutTime*
- *UserIdentifier*
- The version-2 fields (if they were present) are removed from the MQMD, and moved into an MQMDE structure, if one or more of the version-2 fields has a nondefault value.

Putting messages on remote queues: When an application puts a message on a remote queue (either by specifying the name of the remote queue directly, or by using a local definition of the remote queue), the local queue manager:

- Creates an MQXQH structure containing the embedded message descriptor
- Appends an MQMDE if one is needed and is not already present
- Appends the application message data
- Places the message on an appropriate transmission queue

Putting messages directly on transmission queues: An application can also put a message directly on a transmission queue. In this case the application must prefix the application message data with an MQXQH structure, and initialize the fields with appropriate values. In addition, the *Format* field in the *MsgDesc* parameter of the MQPUT or MQPUT1 call must have the value MQFMT_XMIT_Q_HEADER.

Character data in the MQXQH structure created by the application must be in the character set of the local queue manager (defined by the *CodedCharSetId* queue-manager attribute), and integer data must be in the native machine encoding. In addition, character data in the MQXQH structure must be padded with blanks to the defined length of the field; the data must not be ended prematurely by using a null character, because the queue manager does not convert the null and subsequent characters to blanks in the MQXQH structure.

However, the queue manager does not check that an MQXQH structure is present, or that valid values have been specified for the fields.

Getting messages from transmission queues: Applications that get messages from a transmission queue must process the information in the MQXQH structure in an appropriate fashion. The presence of the MQXQH structure at the beginning of the application message data is indicated by the value MQFMT_XMIT_Q_HEADER being returned in the *Format* field in the *MsgDesc* parameter of the MQGET call. The values returned in the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter indicate the character set and encoding of the character and integer data in the MQXQH structure, respectively. The character set and encoding of the application message data are defined by the *CodedCharSetId* and *Encoding* fields in the embedded message descriptor.

Fields for MQXQH

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

MsgDesc (MQMD1)

This is the embedded message descriptor, and is a close copy of the message descriptor MQMD that was specified as the *MsgDesc* parameter on the MQPUT or MQPUT1 call when the message was originally put to the remote queue.

Note: This is a version-1 MQMD.

The initial values of the fields in this structure are the same as those in the MQMD structure.

RemoteQMgrName (MQCHAR48)

This is the name of the queue manager or queue-sharing group that owns the queue that is the apparent eventual destination for the message.

If the message is a distribution-list message, *RemoteQMgrName* is blank.

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

RemoteQName (MQCHAR48)

This is the name of the message queue that is the apparent eventual destination for the message (this might prove not to be the eventual destination if, for example, this queue is defined at *RemoteQMgrName* to be a local definition of another remote queue).

If the message is a distribution-list message (that is, the *Format* field in the embedded message descriptor is MQFMT_DIST_HEADER), *RemoteQName* is blank.

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

StrucId (MQCHAR4)

This is the structure identifier. The value must be:

MQXQH_STRUC_ID

Identifier for transmission-queue header structure.

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

The initial value of this field is MQXQH_STRUC_ID.

Version (MQLONG)

This is the structure version number. The value must be:

MQXQH_VERSION_1

Version number for transmission-queue header structure.

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

MQXQH_CURRENT_VERSION

Current version of transmission-queue header structure.

The initial value of this field is MQXQH_VERSION_1.

Initial values and language declarations for MQXQH

Table 85. Initial values of fields in MQXQH for MQXQH

Field name	Name of constant	Value of constant
<i>StrucId</i>	MQXQH_STRUC_ID	'XQHb'
<i>Version</i>	MQXQH_VERSION_1	1
<i>RemoteQName</i>	None	Null string or blanks
<i>RemoteQMgrName</i>	None	Null string or blanks
<i>MsgDesc</i>	Same names and values as MQMD; see Table 47 on page 235	–
<p>Notes:</p> <ol style="list-style-type: none"> 1. The symbol b represents a single blank character. 2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages. 3. In the C programming language, the macro variable MQXQH_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure: <pre>MQXQH MyXQH = {MQXQH_DEFAULT};</pre> 		

C declaration

```
typedef struct tagMQXQH MQXQH;
struct tagMQXQH {
    MQCHAR4   StrucId;           /* Structure identifier */
    MQLONG    Version;          /* Structure version number */
    MQCHAR48  RemoteQName;      /* Name of destination queue */
    MQCHAR48  RemoteQMgrName;   /* Name of destination queue manager */
    MQMD1     MsgDesc;          /* Original message descriptor */
};
```

COBOL declaration

```
** MQXQH structure
10 MQXQH.
** Structure identifier
15 MQXQH-STRUCID PIC X(4).
** Structure version number
15 MQXQH-VERSION PIC S9(9) BINARY.
** Name of destination queue
15 MQXQH-REMOTEQNAME PIC X(48).
** Name of destination queue manager
15 MQXQH-REMOTEQMGRNAME PIC X(48).
** Original message descriptor
15 MQXQH-MSGDESC.
** Structure identifier
20 MQXQH-MSGDESC-STRUCID PIC X(4).
** Structure version number
20 MQXQH-MSGDESC-VERSION PIC S9(9) BINARY.
** Report options
20 MQXQH-MSGDESC-REPORT PIC S9(9) BINARY.
** Message type
20 MQXQH-MSGDESC-MSGTYPE PIC S9(9) BINARY.
** Expiry time
20 MQXQH-MSGDESC-EXPIRY PIC S9(9) BINARY.
** Feedback or reason code
20 MQXQH-MSGDESC-FEEDBACK PIC S9(9) BINARY.
** Numeric encoding of message data
20 MQXQH-MSGDESC-ENCODING PIC S9(9) BINARY.
```

```

**      Character set identifier of message data
20 MQXQH-MSGDESC-CODEDCHARSETID  PIC S9(9) BINARY.
**      Format name of message data
20 MQXQH-MSGDESC-FORMAT          PIC X(8).
**      Message priority
20 MQXQH-MSGDESC-PRIORITY        PIC S9(9) BINARY.
**      Message persistence
20 MQXQH-MSGDESC-PERSISTENCE     PIC S9(9) BINARY.
**      Message identifier
20 MQXQH-MSGDESC-MSGID           PIC X(24).
**      Correlation identifier
20 MQXQH-MSGDESC-CORRELID        PIC X(24).
**      Backout counter
20 MQXQH-MSGDESC-BACKOUTCOUNT   PIC S9(9) BINARY.
**      Name of reply-to queue
20 MQXQH-MSGDESC-REPLYTOQ        PIC X(48).
**      Name of reply queue manager
20 MQXQH-MSGDESC-REPLYTOQMGR     PIC X(48).
**      User identifier
20 MQXQH-MSGDESC-USERIDENTIFIER  PIC X(12).
**      Accounting token
20 MQXQH-MSGDESC-ACCOUNTINGTOKEN PIC X(32).
**      Application data relating to identity
20 MQXQH-MSGDESC-APPLIDENTITYDATA PIC X(32).
**      Type of application that put the message
20 MQXQH-MSGDESC-PUTAPPLTYPE     PIC S9(9) BINARY.
**      Name of application that put the message
20 MQXQH-MSGDESC-PUTAPPLNAME     PIC X(28).
**      Date when message was put
20 MQXQH-MSGDESC-PUTDATE         PIC X(8).
**      Time when message was put
20 MQXQH-MSGDESC-PUTTIME         PIC X(8).
**      Application data relating to origin
20 MQXQH-MSGDESC-APPLORIGINDATA  PIC X(4).

```

PL/I declaration

```

dcl
1 MQXQH based,
3 StrucId          char(4),          /* Structure identifier */
3 Version          fixed bin(31),    /* Structure version number */
3 RemoteQName      char(48),         /* Name of destination queue */
3 RemoteQMgrName   char(48),         /* Name of destination queue
                                     manager */
3 MsgDesc,        /* Original message descriptor */
5 StrucId          char(4),          /* Structure identifier */
5 Version          fixed bin(31),    /* Structure version number */
5 Report           fixed bin(31),    /* Report options */
5 MsgType          fixed bin(31),    /* Message type */
5 Expiry           fixed bin(31),    /* Expiry time */
5 Feedback         fixed bin(31),    /* Feedback or reason code */
5 Encoding         fixed bin(31),    /* Numeric encoding of message
                                     data */
5 CodedCharSetId   fixed bin(31),    /* Character set identifier of
                                     message data */
5 Format            char(8),          /* Format name of message data */
5 Priority          fixed bin(31),    /* Message priority */
5 Persistence      fixed bin(31),    /* Message persistence */
5 MsgId            char(24),         /* Message identifier */
5 CorrelId         char(24),         /* Correlation identifier */
5 BackoutCount     fixed bin(31),    /* Backout counter */
5 ReplyToQ         char(48),         /* Name of reply-to queue */
5 ReplyToQMgr      char(48),         /* Name of reply queue manager */
5 UserIdentifier   char(12),         /* User identifier */
5 AccountingToken  char(32),         /* Accounting token */
5 ApplIdentityData char(32),         /* Application data relating to
                                     identity */
5 PutApplType      fixed bin(31),    /* Type of application that put the

```

```

                    message */
5 PutAppName      char(28), /* Name of application that put the
                           message */
5 PutDate         char(8),  /* Date when message was put */
5 PutTime        char(8),  /* Time when message was put */
5 ApplOriginData char(4);  /* Application data relating to
                           origin */

```

System/390 assembler declaration

```

MQXQH              DSECT
MQXQH_STRUCID     DS   CL4  Structure identifier
MQXQH_VERSION     DS   F    Structure version number
MQXQH_REMOTEQNAME DS   CL48 Name of destination queue
MQXQH_REMOTEQMGRNAME DS  CL48 Name of destination queue
*                manager
MQXQH_MSGDESC     DS   0F   Force fullword alignment
MQXQH_MSGDESC_STRUCID DS  CL4  Structure identifier
MQXQH_MSGDESC_VERSION DS   F    Structure version number
MQXQH_MSGDESC_REPORT DS   F    Report options
MQXQH_MSGDESC_MSGTYPE DS   F    Message type
MQXQH_MSGDESC_EXPIRY DS   F    Expiry time
MQXQH_MSGDESC_FEEDBACK DS  F    Feedback or reason code
MQXQH_MSGDESC_ENCODING DS  F    Numeric encoding of message
*                data
MQXQH_MSGDESC_CODEDCHARSETID DS  F    Character set identifier of
*                message data
MQXQH_MSGDESC_FORMAT DS  CL8   Format name of message data
MQXQH_MSGDESC_PRIORITY DS   F    Message priority
MQXQH_MSGDESC_PERSISTENCE DS  F    Message persistence
MQXQH_MSGDESC_MSGID DS   XL24  Message identifier
MQXQH_MSGDESC_CORRELID DS  XL24  Correlation identifier
MQXQH_MSGDESC_BACKOUTCOUNT DS  F    Backout counter
MQXQH_MSGDESC_REPLYTOQ DS  CL48  Name of reply-to queue
MQXQH_MSGDESC_REPLYTOQMGR DS  CL48  Name of reply queue manager
MQXQH_MSGDESC_USERIDENTIFIER DS  CL12  User identifier
MQXQH_MSGDESC_ACCOUNTINGTOKEN DS  XL32  Accounting token
MQXQH_MSGDESC_APPLIDENTITYDATA DS  CL32  Application data relating to
*                identity
MQXQH_MSGDESC_PUTAPPLTYPE DS   F    Type of application that put
*                the message
MQXQH_MSGDESC_PUTAPPLNAME DS  CL28  Name of application that put
*                the message
MQXQH_MSGDESC_PUTDATE DS   CL8   Date when message was put
MQXQH_MSGDESC_PUTTIME DS   CL8   Time when message was put
MQXQH_MSGDESC_APPLORIGINDATA DS  CL4  Application data relating to
*                origin
MQXQH_MSGDESC_LENGTH EQU  *-MQXQH_MSGDESC
ORG  MQXQH_MSGDESC
MQXQH_MSGDESC_AREA DS   CL(MQXQH_MSGDESC_LENGTH)
*
MQXQH_LENGTH EQU  *-MQXQH
ORG  MQXQH
MQXQH_AREA DS   CL(MQXQH_LENGTH)

```

Visual Basic declaration

```

Type MQXQH
  StrucId      As String*4  'Structure identifier'
  Version     As Long      'Structure version number'
  RemoteQName As String*48  'Name of destination queue'
  RemoteQMgrName As String*48 'Name of destination queue manager'
  MsgDesc     As MQMD1     'Original message descriptor'
End Type

```

Chapter 2. Function calls

Call descriptions

This part of the book describes the MQI calls:

- MQBACK – Back out
- MQBEGIN – Begin unit of work
- MQBUFMH – Convert buffer into message handle
- MQCB – Manage callback
- MQCB_FUNCTION – Callback function
- MQCLOSE – Close object
- MQCMIT – Commit
- MQCONN – Connect to queue manager
- MQCONNX – Connect to queue manager with options
- MQCRTMH – Create message handle
- MQCTL – Control callback
- MQDISC – Disconnect from queue manager
- MQDLTMH – Delete message handle
- MQDLTMP – Delete message property
- MQGET – Get message
- MQINQ – Inquire about object attributes
- MQINQMP – Inquire message property
- MQMHBUF – Convert message handle into buffer
- MQOPEN – Open object
- MQPUT – Put message
- MQPUT1 – Put one message
- MQSET – Set object attributes
- MQSETMP – Set message handle property
- MQSTAT – Retrieve status information
- MQSUB – Register subscription
- MQSUBRQ – Subscription request

Online help on the UNIX platforms, in the form of *man* pages, is available for these calls.

Note: The calls associated with data conversion, MQXCNVC and MQ_DATA_CONV_EXIT, are in Chapter 8, “Data conversion,” on page 677.

Conventions used in the call descriptions

For each call, this collection of topics gives a description of the parameters and usage of the call in a format that is independent of programming language. This is followed by typical invocations of the call, and typical declarations of its parameters, in each of the supported programming languages.

The description of each call contains the following sections:

Call name

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

Parameters

For each parameter, the name is followed by its data type in parentheses () and one of the following:

input You supply information in the parameter when you make the call.

output

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

input/output

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

For example:

Compcode (MQLONG) — output

In some cases, the data type is a structure. In all cases, there is more information about the data type or structure in “Elementary data types” on page 1.

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 4, “Return codes,” on page 661.

Usage notes

Additional information about the call, describing how to use it and any restrictions on its use.

Assembler language invocation

Typical invocation of the call, and declaration of its parameters, in assembler language.

C invocation

Typical invocation of the call, and declaration of its parameters, in C.

COBOL invocation

Typical invocation of the call, and declaration of its parameters, in COBOL.

PL/I invocation

Typical invocation of the call, and declaration of its parameters, in PL/I.

All parameters are passed by reference.

Visual Basic invocation

Typical invocation of the call, and declaration of its parameters, in Visual Basic.

Other notation conventions are:

Constants

Names of constants are shown in uppercase; for example, MQOO_OUTPUT. A set of constants having the same prefix is shown like this: MQIA_*. See *WebSphere MQ Constants* for the value of a constant.

Arrays

In some calls, parameters are arrays of character strings whose size is not fixed. In the descriptions of these parameters, a lowercase *n* represents a numeric constant. When you code the declaration for that parameter, replace the *n* with the numeric value that you require.

Using the calls in the C language

Parameters that are *input only* and of type MQHCONN, MQHOBJ, MQHMSG, or MQLONG are passed by value. For all other parameters, the *address* of the parameter is passed by value.

You do not need to specify all parameters that are passed by address every time that you invoke a function. Where you do not need a particular parameter, specify a null pointer as the parameter on the function invocation, in place of the address of parameter data. Parameters for which this is possible are identified in the call descriptions.

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

Declaring the Buffer parameter

The MQGET, MQPUT, and MQPUT1 calls each have one parameter that has an undefined data type: the *Buffer* parameter. Use this parameter to send and receive the application's message data.

Parameters of this sort are shown in the C examples as arrays of MQBYTE. You can declare the parameters in this way, but it is usually more convenient to declare them as the particular structure that describes the layout of the data in the message. The function prototype declares the parameter as a pointer-to-void, so that you can specify the address of any sort of data as the parameter on the call invocation.

Pointer-to-void is a pointer to data of undefined format. It is defined as:

```
typedef void *PMQVOID;
```

MQBACK – Back out changes

The MQBACK call indicates to the queue manager that all the message gets and puts that have occurred since the last syncpoint are to be backed out.

Messages put as part of a unit of work are deleted; messages retrieved as part of a unit of work are reinstated on the queue.

- On z/OS, this call is used only by batch programs (including IMS batch DL/I programs).
- On i5/OS, this call is not supported for applications running in compatibility mode.

Syntax for MQBACK

MQBACK (*Hconn*, *CompCode*, *Reason*)

Parameters for MQBACK

The MQBACK call has the following parameters.

Hconn (MQHCONN) – input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

CompCode (MQLONG) – output

The completion code; it is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

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

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_SERV_LOAD_ERROR

(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR

(2374, X'946') API exit failed.

MQRC_ASID_MISMATCH

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

MQRC_CALL_IN_PROGRESS

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

MQRC_CF_STRUC_IN_USE

(2346, X'92A') Coupling-facility structure in use.

MQRC_CONNECTION_BROKEN

(2009, X'7D9') Connection to queue manager lost.

MQRC_ENVIRONMENT_ERROR

(2012, X'7DC') Call not valid in environment.

MQRC_HCONN_ERROR

(2018, X'7E2') Connection handle not valid.

MQRC_OBJECT_DAMAGED

(2101, X'835') Object damaged.

MQRC_OUTCOME_MIXED

(2123, X'84B') Result of commit or back-out operation is mixed.

MQRC_Q_MGR_STOPPING

(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM

(2102, X'836') Insufficient system resources available.

MQRC_STORAGE_MEDIUM_FULL
(2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

See Chapter 4, "Return codes," on page 661 for more details.

Usage notes for MQBACK

Usage notes for MQBACK.

1. You can use this call only when the queue manager itself coordinates the unit of work. This can be:
 - A local unit of work, where the changes affect only MQ resources.
 - A global unit of work, where the changes can affect resources belonging to other resource managers, as well as affecting MQ resources.

For further details about local and global units of work, see "MQBEGIN – Begin unit of work" on page 401.

2. In environments where the queue manager does not coordinate the unit of work, use the appropriate back-out call instead of MQBACK. The environment might also support an implicit back out caused by the application terminating abnormally.
 - On z/OS, use the following calls:
 - Batch programs (including IMS batch DL/I programs) can use the MQBACK call if the unit of work affects only MQ resources. However, if the unit of work affects both MQ resources and resources belonging to other resource managers (for example, DB2[®]), use the SRRBACK call provided by the z/OS Recoverable Resource Service (RRS). The SRRBACK call backs out changes to resources belonging to the resource managers that have been enabled for RRS coordination.
 - CICS applications must use the EXEC CICS SYNCPOINT ROLLBACK command to back out the unit of work. Do not use the MQBACK call for CICS applications.
 - IMS applications (other than batch DL/I programs) must use IMS calls such as R0LB to back out the unit of work. Do not use the MQBACK call for IMS applications (other than batch DL/I programs).
 - On i5/OS, use this call for local units of work coordinated by the queue manager. This means that a commitment definition must not exist at job level, that is, the STRCMTCTL command with the CMTSCOPE(*JOB) parameter must not have been issued for the job.
3. If an application ends with uncommitted changes in a unit of work, the disposition of those changes depends on whether the application ends normally or abnormally. See the usage notes in "MQDISC – Disconnect queue manager" on page 457 for further details.
4. When an application puts or gets messages in groups or segments of logical messages, the queue manager retains information relating to the message group and logical message for the last successful MQPUT and MQGET calls. This information is associated with the queue handle, and includes such things as:
 - The values of the *GroupId*, *MsgSeqNumber*, *Offset*, and *MsgFlags* fields in MQMD.
 - Whether the message is part of a unit of work.

- For the MQPUT call: whether the message is persistent or nonpersistent.

The queue manager keeps *three* sets of group and segment information, one set for each of the following:

- The last successful MQPUT call (this can be part of a unit of work).
 - The last successful MQGET call that removed a message from the queue (this can be part of a unit of work).
 - The last successful MQGET call that browsed a message on the queue (this *cannot* be part of a unit of work).
5. The information associated with the MQGET call is restored to the value that it had before the first successful MQGET call for that queue handle in the current unit of work.

Queues that were updated by the application after the unit of work started, but outside the scope of the unit of work, do not have their group and segment information restored if the unit of work is backed out.

Restoring the group and segment information to its previous value when a unit of work is backed out allows the application to spread a large message group or large logical message consisting of many segments across several units of work, and to restart at the correct point in the message group or logical message if one of the units of work fails. Using several units of work might be advantageous if the local queue manager has only limited queue storage. However, the application must maintain sufficient information to be able to restart putting or getting messages at the correct point in the event that a system failure occurs. For details of how to restart at the correct point after a system failure, see the MQPMO_LOGICAL_ORDER option described in “MQPMO – Put-message options” on page 272, and the MQGMO_LOGICAL_ORDER option described in “MQGMO – Get-message options” on page 126.

The remaining usage notes apply only when the queue manager coordinates the units of work:

- 6.
7. A unit of work has the same scope as a connection handle. All MQ calls that affect a particular unit of work must be performed using the same connection handle. Calls issued using a different connection handle (for example, calls issued by another application) affect a different unit of work. See the *Hconn* parameter described in “MQCONN – Connect queue manager” on page 435 for information about the scope of connection handles.
8. Only messages that were put or retrieved as part of the current unit of work are affected by this call.
9. A long-running application that issues MQGET, MQPUT, or MQPUT1 calls within a unit of work, but that never issues a commit or backout call, can fill queues with messages that are not available to other applications. To guard against this possibility, the administrator must set the *MaxUncommittedMsgs* queue-manager attribute to a value that is low enough to prevent runaway applications filling the queues, but high enough to allow the expected messaging applications to work correctly.

Language invocations for MQBACK

The MQBACK call is supported in the programming languages shown below.

C invocation

```
MQBACK (Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn; /* Connection handle */
MQLONG CompCode; /* Completion code */
MQLONG Reason; /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQBACK' USING HCONN, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN PIC S9(9) BINARY.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON PIC S9(9) BINARY.
```

PL/I invocation

```
call MQBACK (Hconn, CompCode, Reason);
```

Declare the parameters as follows:

```
dc1 Hconn fixed bin(31); /* Connection handle */
dc1 CompCode fixed bin(31); /* Completion code */
dc1 Reason fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
CALL MQBACK,(HCONN,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN DS F Connection handle
COMPCODE DS F Completion code
REASON DS F Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQBACK Hconn, CompCode, Reason
```

Declare the parameters as follows:

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

MQBEGIN – Begin unit of work

The MQBEGIN call begins a unit of work that is coordinated by the queue manager, and that can involve external resource managers.

- This call is supported in the following environments: AIX, HP-UX, i5/OS, Solaris, Linux, Windows.

Syntax for MQBEGIN

```
MQBEGIN (Hconn, BeginOptions, CompCode, Reason)
```


Parameters for MQBEGIN

The MQBEGIN call has the following parameters.

Hconn (MQHCONN) – input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

Hconn must be a nonshared connection handle. If a shared connection handle is specified, the call fails with reason code MQRC_HCONN_ERROR. See the description of the MQCNO_HANDLE_SHARE_* options in “MQCNO – Connect options” on page 77 for more information about shared and nonshared handles.

BeginOptions (MQBO) – input/output

These are options that control the action of MQBEGIN, as described in see “MQBO – Begin options” on page 35.

If no options are required, programs written in C or S/390® assembler can specify a null parameter address, instead of specifying the address of an MQBO structure.

CompCode (MQLONG) – output

This is the completion code; it is one of the following:

MQCC_OK

Successful completion.

MQCC_WARNING

Warning (partial completion).

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

This is the reason code qualifying *CompCode*. If *CompCode* is MQCC_OK:

MQRC_NONE

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

If *CompCode* is MQCC_WARNING:

MQRC_NO_EXTERNAL_PARTICIPANTS

(2121, X'849') No participating resource managers registered.

MQRC_PARTICIPANT_NOT_AVAILABLE

(2122, X'84A') Participating resource manager not available.

If *CompCode* is MQCC_FAILED:

MQRC_API_EXIT_ERROR

(2374, X'946') API exit failed.

MQRC_BO_ERROR

(2134, X'856') Begin-options structure not valid.

MQRC_CALL_IN_PROGRESS

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

- MQRC_CONNECTION_BROKEN**
(2009, X'7D9') Connection to queue manager lost.
- MQRC_ENVIRONMENT_ERROR**
(2012, X'7DC') Call not valid in environment.
- MQRC_HCONN_ERROR**
(2018, X'7E2') Connection handle not valid.
- MQRC_OPTIONS_ERROR**
(2046, X'7FE') Options not valid or not consistent.
- MQRC_Q_MGR_STOPPING**
(2162, X'872') Queue manager shutting down.
- MQRC_RESOURCE_PROBLEM**
(2102, X'836') Insufficient system resources available.
- MQRC_STORAGE_NOT_AVAILABLE**
(2071, X'817') Insufficient storage available.
- MQRC_UNEXPECTED_ERROR**
(2195, X'893') Unexpected error occurred.
- MQRC_UOW_IN_PROGRESS**
(2128, X'850') Unit of work already started.

For more information on these reason codes, see

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

Usage notes for MQBEGIN

Consider these points when using MQBEGIN.

1. Use the MQBEGIN call to start a unit of work that is coordinated by the queue manager and that might involve changes to resources owned by other resource managers. The queue manager supports three types of unit-of-work:
 - **Queue-manager-coordinated local unit of work:** This is a unit of work in which the queue manager is the only resource manager participating, and so the queue manager acts as the unit-of-work coordinator.
 - To start this type of unit of work, specify the MQPMO_SYNCPOINT or MQGMO_SYNCPOINT option on the first MQPUT, MQPUT1, or MQGET call in the unit of work.
 - To commit or back out this type of unit of work, use the MQCMIT or MQBACK call.
 - **Queue-manager-coordinated global unit of work:** This is a unit of work in which the queue manager acts as the unit-of-work coordinator, both for MQ resources *and* for resources belonging to other resource managers. Those resource managers cooperate with the queue manager to ensure that all changes to resources in the unit of work are committed or backed out together.
 - To start this type of unit of work, use the MQBEGIN call.
 - To commit or back out this type of unit of work, use the MQCMIT and MQBACK calls.
 - **Externally-coordinated global unit of work:** This is a unit of work in which the queue manager is a participant, but the queue manager does not act as

the unit-of-work coordinator. Instead, there is an external unit-of-work coordinator with which the queue manager cooperates.

- To start this type of unit of work, use the relevant call provided by the external unit-of-work coordinator.

If the MQBEGIN call is used to try to start the unit of work, the call fails with reason code MQRC_ENVIRONMENT_ERROR.

- To commit or back out this type of unit of work, use the commit and back-out calls provided by the external unit-of-work coordinator.

If you use the MQCMIT or MQBACK call to commit or back out the unit of work, the call fails with reason code MQRC_ENVIRONMENT_ERROR.

2. If the application ends with uncommitted changes in a unit of work, the disposition of those changes depends on whether the application ends normally or abnormally. See the usage notes in “MQDISC – Disconnect queue manager” on page 457 for further details.
3. An application can participate in only one unit of work at a time. The MQBEGIN call fails with reason code MQRC_UOW_IN_PROGRESS if there is already a unit of work in existence for the application, regardless of which type of unit of work it is.
4. The MQBEGIN call is not valid in an MQ client environment. An attempt to use the call fails with reason code MQRC_ENVIRONMENT_ERROR.
5. When the queue manager is acting as the unit-of-work coordinator for global units of work, the resource managers that can participate in the unit of work are defined in the queue manager’s configuration file.
6. On i5/OS, the three types of unit of work are supported as follows:
 - **Queue-manager-coordinated local unit of work** can be used only when a commitment definition does not exist at the job level, that is, the STRCMTCTL command with the CMTSCOPE(*JOB) parameter must not have been issued for the job.
 - **Queue-manager-coordinated global unit of work** is not supported.
 - **Externally-coordinated global unit of work** can be used only when a commitment definition exists at job level, that is, the STRCMTCTL command with the CMTSCOPE(*JOB) parameter must have been issued for the job. If this has been done, the i5/OS COMMIT and ROLLBACK operations apply to MQ resources as well as to resources belonging to other participating resource managers.

Language invocations for MQBEGIN

The MQBEGIN call is supported in the programming languages shown below.

C invocation

```
MQBEGIN (Hconn, &BeginOptions, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;          /* Connection handle */
MQBO     BeginOptions;   /* Options that control the action of MQBEGIN */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQBEGIN' USING HCONN, BEGINOPTIONS, COMPCODE, REASON.
```

Declare the parameters as follows:

```

** Connection handle
01 HCONN          PIC S9(9) BINARY.
** Options that control the action of MQBEGIN
01 BEGINOPTIONS.
   COPY CMQBOV.
** Completion code
01 COMPCODE       PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON         PIC S9(9) BINARY.

```

PL/I invocation

```
call MQBEGIN (Hconn, BeginOptions, CompCode, Reason);
```

Declare the parameters as follows:

```

dcl Hconn          fixed bin(31); /* Connection handle */
dcl BeginOptions  like MQBO;     /* Options that control the action of
                                MQBEGIN */
dcl CompCode      fixed bin(31); /* Completion code */
dcl Reason        fixed bin(31); /* Reason code qualifying CompCode */

```

Visual Basic invocation

```
MQBEGIN Hconn, BeginOptions, CompCode, Reason
```

Declare the parameters as follows:

```

Dim Hconn          As Long 'Connection handle'
Dim BeginOptions  As MQBO 'Options that control the action of MQBEGIN'
Dim CompCode      As Long 'Completion code'
Dim Reason        As Long 'Reason code qualifying CompCode'

```

MQBUFMH - Convert buffer into message handle

The MQBUFMH function call converts a buffer into a message handle and is the inverse of the MQMHBUF call.

This call takes a message descriptor and MQRFH2 properties in the buffer and makes them available through a message handle. The MQRFH2 properties in the message data are, optionally, removed. The *Encoding*, *CodedCharSetId*, and *Format* fields of the message descriptor are updated, if necessary, to correctly describe the contents of the buffer after the properties have been removed.

Syntax for MQBUFMH

```
MQBUFMH (Hconn, Hmsg, BufMsgHOpts, MsgDesc, Buffer, BufferLength, DataLength, CompCode, Reason)
```

Parameters for MQBUFMH

The MQBUFMH call has the following parameters.

Hconn (MQHCONN) – input

This handle represents the connection to the queue manager. The value of *Hconn* must match the connection handle that was used to create the message handle specified in the *Hmsg* parameter.

If the message handle was created using MQHC_UNASSOCIATED_HCONN, a valid connection must be established on the thread converting a buffer into a message handle. If a valid connection is not established, the call fails with MQRC_CONNECTION_BROKEN.

Hmsg (MQHMSG) – input

This is the message handle for which a buffer is required. The value was returned by a previous MQCRTMH call.

BufMsgHOpts (MQBMHO) – input

The MQBMHO structure allows applications to specify options that control how message handles are produced from buffers.

See “MQBMHO – Buffer to message handle options” on page 33 for details.

MsgDesc (MQMD) – input/output

The *MsgDesc* structure contains the message descriptor properties and describes the contents of the buffer area.

On output from the call, the properties are optionally removed from the buffer area and, in this case, the message descriptor is updated to correctly describe the buffer area.

Data in this structure must be in the character set and encoding of the application.

BufferLength (MQLONG) - input

BufferLength is the length of the Buffer area, in bytes.

A *BufferLength* of zero bytes is valid, and indicates that the buffer area contains no data.

Buffer (MQBYTEExBufferLength) - input/output

Buffer defines the area containing the message buffer. For most data, you should align the buffer on a 4-byte boundary.

If *Buffer* contains character or numeric data, set the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter to the values appropriate to the data; this enables the data to be converted, if necessary.

If properties are found in the message buffer they are optionally removed; they later become available from the message handle on return from the call.

In the C programming language, the parameter is declared as a pointer-to-void, which means the address of any type of data can be specified as the parameter.

If the *BufferLength* parameter is zero, *Buffer* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler can be null.

DataLength (MQLONG) - output

DataLength is the length, in bytes, of the buffer which might have the properties removed.

CompCode (MQLONG) – output

The completion code; it is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

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

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE

(2204, X'089C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR

(2130, X'852') Unable to load adapter service module.

MQRC_ASID_MISMATCH

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

MQRC_BMHO_ERROR

(2489, X'09B9') Buffer to message handle options structure not valid.

MQRC_BUFFER_ERROR

(2004, X'07D4') Buffer parameter not valid.

MQRC_BUFFER_LENGTH_ERROR

(2005, X'07D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS

(2219, X'08AB') MQI call entered before previous call completed.

MQRC_CONNECTION_BROKEN

(2009, X'07D9') Connection to queue manager lost.

MQRC_HMSG_ERROR

(2460, X'099C') Message handle not valid.

MQRC_MD_ERROR

(2026, X'07EA') Message descriptor not valid.

MQRC_MSG_HANDLE_IN_USE

(2499, X'09C3') Message handle already in use.

MQRC_OPTIONS_ERROR

(2046, X'07FE') Options not valid or not consistent.

MQRC_RFH_ERROR

(2334, X'091E') MQRFH2 structure not valid.

MQRC_RFH_FORMAT_ERROR

(2421, X'0975') An MQRFH2 folder containing properties could not be parsed.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

Usage notes for MQBUFMH

MQBUFMH calls cannot be intercepted by API exits – a buffer is converted into a message handle in the application space; the call does not reach the queue manager.

Language invocations for MQBUFMH

The MQBUFMH call is supported in the programming languages shown below.

C invocation

```
MQBUFMH (Hconn, Hmsg, &BufMsgHOpts, &MsgDesc, BufferLength, Buffer,  
         &DataLength, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;      /* Connection handle */  
MQHMSG  Hmsg;       /* Message handle */  
MQBMHO  BufMsgHOpts; /* Options that control the action of MQBUFMH */  
MQMD    MsgDesc;    /* Message descriptor */  
MQLONG  BufferLength; /* Length in bytes of the Buffer area */  
MQBYTE  Buffer[n];   /* Area to contain the message buffer */  
MQLONG  DataLength; /* Length of the output buffer */  
MQLONG  CompCode;   /* Completion code */  
MQLONG  Reason;     /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQBUFMH' USING HCONN, HMSG, BUFMSGHOPTS, MSGDESC, BUFFERLENGTH,  
                   BUFFER, DATALENGTH, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle  
01 HCONN          PIC S9(9) BINARY.  
** Message handle  
01 HMSG           PIC S9(18) BINARY.  
** Options that control the action of MQBUFMH  
01 BUFMSGHOPTS.  
   COPY CMQBMHOV.  
** Message descriptor  
01 MSGDESC.  
   COPY CMQMD.  
** Length in bytes of the Buffer area  
01 BUFFERLENGTH  PIC S9(9) BINARY.  
** Area to contain the message buffer  
01 BUFFER        PIC X(n).  
** Length of the output buffer  
01 DATALENGTH   PIC S9(9) BINARY.  
** Completion code  
01 COMPCODE      PIC S9(9) BINARY.  
** Reason code qualifying COMPCODE  
01 REASON        PIC S9(9) BINARY.
```

PL/I invocation

```
call MQBUFMH (Hconn, Hmsg, BufMsgHOpts, MsgDesc, BufferLength, Buffer,  
             DataLength, CompCode, Reason);
```

Declare the parameters as follows:


```

dc1 Hconn          fixed bin(31); /* Connection handle */
dc1 Hmsg           fixed bin(63); /* Message handle */
dc1 BufMsgHOpts   like MQBMHO; /* Options that control the action of
                               MQBUFMH */
dc1 MsgDesc       like MQMD; /* Message descriptor */
dc1 BufferLength   fixed bin(31); /* Length in bytes of the Buffer area */
dc1 Buffer         char(n); /* Area to contain the message buffer */
dc1 DataLength    fixed bin(31); /* Length of the output buffer */
dc1 CompCode      fixed bin(31); /* Completion code */
dc1 Reason        fixed bin(31); /* Reason code qualifying CompCode */

```

System/390 assembler invocation

```

CALL MQBUFMH, (HCONN,HMSG,BUFMSGHOPTS,MSGDESC,BUFFERLENGTH,BUFFER,
              DATALENGTH,COMPCODE,REASON)

```

Declare the parameters as follows:

HCONN	DS	F	Connection handle
HMSG	DS	D	Message handle
BUFMSGHOPTS	CMQBMHOA	,	Options that control the action of MQBUFMH
MSGDESC	CMQMDA	,	Message descriptor
BUFFERLENGTH	DS	F	Length in bytes of the BUFFER area
BUFFER	DS	CL(n)	Area to contain the properties
DATALENGTH	DS	F	Length of the output buffer
COMPCODE	DS	F	Completion code
REASON	DS	F	Reason code qualifying COMPCODE

MQCB – Manage callback

Manage callback function

The MQCB call reregisters a callback for the specified object handle and controls activation and changes to the callback.

A callback is a piece of code (specified as either the name of a function that can be dynamically linked or as function pointer) that is called by WebSphere MQ when certain events occur.

To use MQCB and MQCTL on a V7 client you must be connected to a V7 server and the **SHARECNV** parameter of the channel must have a non-zero value.

The types of callback that can be defined are:

Message consumer

A message consumer callback function is called when a message, meeting the selection criteria specified, is available on an object handle.

Only one callback function can be registered against each object handle. If a single queue is to be read with multiple selection criteria then the queue must be opened multiple times and a consumer function registered on each handle.

Event handler

The event handler is called for conditions that affect the whole callback environment.

The function is called when an event condition occurs, for example, a queue manager or connection stopping or quiescing.

The function is not called for conditions that are specific to a single message consumer, for example **MQRC_GET_INHIBITED**; it is called however if a callback function does not end normally.

Syntax for MQCB

Message callback function - syntax

MQCB (*Hconn, Operation, CallbackDesc, Hobj, MsgDesc, GetMsgOpts, CompCode, Reason*)

Parameters for MQCB

The MQCB call has the following parameters. Manage callback function - parameters

Hconn (MQHCONN) – input

Manage callback function - Hconn parameter

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNEX call.

On z/OS for CICS applications, and on i5/OS for applications running in compatibility mode, you can specify the following special value for *MQHC_DEF_HCONN* to use the connection handle associated with this execution unit.

Operation (MQLONG) – input

Manage callback function - Operation parameter

The operation being processed on the callback defined for the specified object handle. You must specify one of the following options; if more than one option is required, the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

MQOP_REGISTER

Define the callback function for the specified object handle. This operation defines the function to be called and the selection criteria to be used.

If a callback function is already defined for the object handle the definition is replaced. If an error is detected whilst replacing the callback, the function is deregistered.

If a callback is registered in the same callback function in which it was previously deregistered, this is treated as a replace operation; any initial or final calls are not invoked.

You can use MQOP_REGISTER in conjunction with MQOP_SUSPEND or MQOP_RESUME.

MQOP_DEREGISTER

Stop the consuming of messages for the object handle and removes the handle from those eligible for a callback.

A callback is automatically deregistered if the associated handle is closed.

If MQOP_DEREGISTER is called from within a consumer, and the callback has a stop call defined, it is invoked upon return from the consumer.

If this operation is issued against an *Hobj* with no registered consumer, the call returns with MQRC_CALLBACK_NOT_REGISTERED.

MQOP_SUSPEND

Suspends the consuming of messages for the object handle.

If this operation is applied to an event handler, the event handler does not get events whilst suspended, and any events missed while in the suspended state are not provided to the operation when it is resumed.

While suspended, the consumer function continues to get the control type callbacks.

MQOP_RESUME

Resume the consuming of messages for the object handle.

If this operation is applied to an event handler, the event handler does not get events whilst suspended, and any events missed while in the suspended state are not provided to the operation when it is resumed.

CallbackDesc (MQCBD) – input

Manage callback function -CallbackDesc parameter

This is a structure that identifies the callback function that is being registered by the application and the options used when registering it.

See MQCBD for details of the structure.

Callback descriptor is required only for the MQOP_REGISTER option; if the descriptor is not required, the parameter address passed can be null.

Hobj (MQHOBJ) – input

Manage callback function -Hobj parameter

This handle represents the access that has been established to the object from which a message is to be consumed. This is a handle that has been returned from a previous MQOPEN or MQSUB call (in the *Hobj* parameter).

Hobj is not required when defining an event handler routine (MQCBT_EVENT_HANDLER) and should be specified as MQHO_NONE.

If this *Hobj* has been returned from an MQOPEN call, the queue must have been opened with one or more of the following options:

- MQOO_INPUT_SHARED
- MQOO_INPUT_EXCLUSIVE
- MQOO_INPUT_AS_Q_DEF
- MQOO_BROWSE

MsgDesc (MQMD) – input

Manage callback function -MsgDesc parameter

This structure describes the attributes of the message required, and the attributes of the message retrieved.

The *MsgDesc* parameter defines the attributes of the messages required by the consumer, and the version of the MQMD to be passed to the message consumer.

The *MsgId*, *CorrelId*, *GroupId*, *MsgSeqNumber*, and *Offset* in the MQMD are used for message selection, depending on the options specified in the *GetMsgOpts* parameter.

The *Encoding* and *CodedCharSetId* are used for message conversion if you specify the MQGMO_CONVERT option.

See MQMD for details.

MsgDesc is used for MQOP_REGISTER and if you require values other than the default for any fields. *MsgDesc* is not used for an event handler.

If the descriptor is not required the parameter address passed can be null.

Note, that if multiple consumers are registered against the same queue with overlapping selectors, the chosen consumer for each message is undefined.

GetMsgOpts (MQGMO) – input

Manage callback function - *GetMsgOpts* parameter

The *GetMsgOpts* parameter controls how the message consumer gets messages. All options of this parameter have meanings as described in “MQGMO – Get-message options” on page 126, when used on an MQGET call, except:

MQGMO_SET_SIGNAL

This option is not permitted.

MQGMO_BROWSE_FIRST, MQGMO_BROWSE_NEXT, MQGMO_MARK_*

The order of messages delivered to a browsing consumer is dictated by the combinations of these options. Significant combinations are:

MQGMO_BROWSE_FIRST

The first message on the queue is delivered repeatedly to the consumer. This is useful when the consumer destructively consumes the message in the callback. Use this option with care.

MQGMO_BROWSE_NEXT

The consumer is given each message on the queue, from the current cursor position until the end of the queue is reached.

MQGMO_BROWSE_FIRST + MQGMO_BROWSE_NEXT

The cursor is reset to the start of the queue. The consumer is then given each message until the cursor reaches the end of the queue.

MQGMO_BROWSE_FIRST + MQGMO_MARK_*

Starting at the beginning of the queue, the consumer is given the first nonmarked message on the queue, which is then marked for this consumer. This combination ensures that the consumer can receive new messages added behind the current cursor point.

MQGMO_BROWSE_NEXT + MQGMO_MARK_*

Starting at the cursor position, the consumer is given the next nonmarked message on the queue, which is then marked for this consumer. Use this combination with care because messages can be added to the queue behind the current cursor position.

MQGMO_BROWSE_FIRST + MQGMO_BROWSE_NEXT + MQGMO_MARK_*

This combination is not permitted. If used the call returns MQRC_OPTIONS_ERROR.

MQGMO_NO_WAIT, MQGMO_WAIT and WaitInterval

These options control how the consumer is invoked.

MQGMO_NO_WAIT

The consumer is never called with MQRC_NO_MSG_AVAILABLE.
The consumer is only called for messages and events.

MQGMO_WAIT with a zero WaitInterval

The MQRC_NO_MSG_AVAILABLE code is passed to the consumer when there are no messages available and either the consumer has just been started or the consumer has been delivered at least one message since the last "no messages" reason code.

This prevents the consumer from polling in a busy loop when a zero wait interval is specified.

MQGMO_WAIT and a positive WaitInterval

The consumer is called after the specified wait interval with reason code MQRC_NO_MSG_AVAILABLE. This call is made regardless of whether any messages have been delivered to the consumer. This allows the user to perform heartbeat or batch type processing.

MQGMO_WAIT and WaitInterval of MQWI_UNLIMITED

This specifies an infinite wait before returning MQRC_NO_MSG_AVAILABLE. The consumer is never called with MQRC_NO_MSG_AVAILABLE.

GetMsgOpts is used only for MQOP_REGISTER and if you require values other than the default for any fields. *GetMsgOpts* is not used for an event handler.

If the *GetMsgOpts* are not required, the parameter address passed can be null. This is the as specifying MQGMO_DEFAULT together with MQGMO_FAIL_IF_QUIESCING.

If a message properties handle is provided in the MQGMO structure, a copy is provided in the MQGMO structure that is passed into the consumer callback. On return from the MQCB call, the application can delete the message properties handle.

CompCode (MQLONG) – output

Manage callback function - CompCode parameter

The completion code; it is one of the following:

MQCC_OK

Successful completion.

MQCC_WARNING

Warning (partial completion).

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Manage callback function - Reason parameter

The reason codes listed below are the ones that the queue manager can return for the *Reason* parameter.

If *CompCode* is MQCC_OK:

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

|
 | If *CompCode* is MQCC_FAILED:

| **MQRC_ADAPTER_NOT_AVAILABLE**
 | (2204, X'89C') Adapter not available.

| **MQRC_ADAPTER_CONV_LOAD_ERROR**
 | (2133, X'855') Unable to load data conversion services modules.

| **MQRC_ADAPTER_SERV_LOAD_ERROR**
 | (2130, X'852') Unable to load adapter service module.

| **MQRC_API_EXIT_ERROR**
 | (2374, X'946') API exit failed.

| **MQRC_API_EXIT_LOAD_ERROR**
 | (2183, X'887') Unable to load API exit.

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

| **MQRC_BUFFER_LENGTH_ERROR**
 | (2005, X'7D5') Buffer length parameter not valid.

| **MQRC_CALL_IN_PROGRESS**
 | (2219, X'8AB') MQI call entered before previous call complete.

| **MQRC_CALLBACK_LINK_ERROR**
 | (2487, X'9B7') Incorrect callback type field.

| **MQRC_CALLBACK_NOT_REGISTERED**
 | (2448, X'990') Unable to deregister, suspend, or resume because there is no
 | registered callback.

| **MQRC_CALLBACK_ROUTINE_ERROR**
 | (2486, X'9B6') Either *CallbackFunction* or *CallbackName* must be specified
 | but not both.

| **MQRC_CALLBACK_TYPE_ERROR**
 | (2483, X'9B3') Incorrect callback type field.

| **MQRC_CBD_OPTIONS_ERROR**
 | (2484, X'9B4') Incorrect MQCBD options field.

| **MQRC_CICS_WAIT_FAILED**
 | (2140, X'85C') Wait request rejected by CICS.

| **MQRC_CONNECTION_BROKEN**
 | (2009, X'7D9') Connection to queue manager lost.

| **MQRC_CONNECTION_NOT_AUTHORIZED**
 | (2217, X'8A9') Not authorized for connection.

| **MQRC_CONNECTION_QUIESCING**
 | (2202, X'89A') Connection quiescing.

| **MQRC_CONNECTION_STOPPING**
 | (2203, X'89B') Connection shutting down.

| **MQRC_CORREL_ID_ERROR**
 | (2207, X'89F') Correlation-identifier error.

| **MQRC_DATA_LENGTH_ERROR**
 | (2010, X'7DA') Data length parameter not valid.

| **MQRC_GET_INHIBITED**
| (2016, X'7E0') Gets inhibited for the queue.

| **MQRC_GLOBAL_UOW_CONFLICT**
| (2351, X'92F') Global units of work conflict.

| **MQRC_GMO_ERROR**
| (2186, X'88A') Get-message options structure not valid.

| **MQRC_HANDLE_IN_USE_FOR_UOW**
| (2353, X'931') Handle in use for global unit of work.

| **MQRC_HCONN_ERROR**
| (2018, X'7E2') Connection handle not valid.

| **MQRC_HOBJ_ERROR**
| (2019, X'7E3') Object handle not valid.

| **MQRC_INCONSISTENT_BROWSE**
| (2259, X'8D3') Inconsistent browse specification.

| **MQRC_INCONSISTENT_UOW**
| (2245, X'8C5') Inconsistent unit-of-work specification.

| **MQRC_INVALID_MSG_UNDER_CURSOR**
| (2246, X'8C6') Message under cursor not valid for retrieval.

| **MQRC_LOCAL_UOW_CONFLICT**
| (2352, X'930') Global unit of work conflicts with local unit of work.

| **MQRC_MATCH_OPTIONS_ERROR**
| (2247, X'8C7') Match options not valid.

| **MQRC_MAX_MSG_LENGTH_ERROR**
| (2485, X'9B4') Incorrect *MaxMsgLength* field.

| **MQRC_MD_ERROR**
| (2026, X'7EA') Message descriptor not valid.

| **MQRC_MODULE_ENTRY_NOT_FOUND**
| (2497, X'9C1') The specified function entry point could not be found in the
| module.

| **MQRC_MODULE_INVALID**
| (2496, X'9C0') Module found, however it is of the wrong type; not 32 bit,
| 64 bit, or a valid dynamic link library.

| **MQRC_MODULE_NOT_FOUND**
| (2495, X'9BF') Module not found in the search path or not authorized to
| load.

| **MQRC_MSG_SEQ_NUMBER_ERROR**
| (2250, X'8CA') Message sequence number not valid.

| **MQRC_MSG_TOKEN_ERROR**
| (2331, X'91B') Use of message token not valid.

| **MQRC_NO_MSG_AVAILABLE**
| (2033, X'7F1') No message available.

| **MQRC_NO_MSG_UNDER_CURSOR**
| (2034, X'7F2') Browse cursor not positioned on message.

| **MQRC_NOT_OPEN_FOR_BROWSE**
| (2036, X'7F4') Queue not open for browse.

| **MQRC_NOT_OPEN_FOR_INPUT**
| (2037, X'7F5') Queue not open for input.

| **MQRC_OBJECT_CHANGED**
| (2041, X'7F9') Object definition changed since opened.

| **MQRC_OBJECT_DAMAGED**
| (2101, X'835') Object damaged.

| **MQRC_OPERATION_ERROR**
| (2206, X'89E') Incorrect operation code on API Call.

| **MQRC_OPTIONS_ERROR**
| (2046, X'7FE') Options not valid or not consistent.

| **MQRC_PAGESET_ERROR**
| (2193, X'891') Error accessing page-set data set.

| **MQRC_Q_DELETED**
| (2052, X'804') Queue has been deleted.

| **MQRC_Q_INDEX_TYPE_ERROR**
| (2394, X'95A') Queue has wrong index type.

| **MQRC_Q_MGR_NAME_ERROR**
| (2058, X'80A') Queue manager name not valid or not known.

| **MQRC_Q_MGR_NOT_AVAILABLE**
| (2059, X'80B') Queue manager not available for connection.

| **MQRC_Q_MGR QUIESCING**
| (2161, X'871') Queue manager quiescing.

| **MQRC_Q_MGR_STOPPING**
| (2162, X'872') Queue manager shutting down.

| **MQRC_RESOURCE_PROBLEM**
| (2102, X'836') Insufficient system resources available.

| **MQRC_SIGNAL_OUTSTANDING**
| (2069, X'815') Signal outstanding for this handle.

| **MQRC_STORAGE_NOT_AVAILABLE**
| (2071, X'817') Insufficient storage available.

| **MQRC_SUPPRESSED_BY_EXIT**
| (2109, X'83D') Call suppressed by exit program.

| **MQRC_SYNCPOINT_LIMIT_REACHED**
| (2024, X'7E8') No more messages can be handled within current unit of
| work.

| **MQRC_SYNCPOINT_NOT_AVAILABLE**
| (2072, X'818') Syncpoint support not available.

| **MQRC_UNEXPECTED_ERROR**
| (2195, X'893') Unexpected error occurred.

| **MQRC_UOW_ENLISTMENT_ERROR**
| (2354, X'932') Enlistment in global unit of work failed.

| **MQRC_UOW_MIX_NOT_SUPPORTED**
| (2355, X'933') Mixture of unit-of-work calls not supported.

| **MQRC_UOW_NOT_AVAILABLE**
| (2255, X'8CF') Unit of work not available for the queue manager to use.

MQRC_WAIT_INTERVAL_ERROR
(2090, X'82A') Wait interval in MQGMO not valid.

MQRC_WRONG_GMO_VERSION
(2256, X'8D0') Wrong version of MQGMO supplied.

MQRC_WRONG_MD_VERSION
(2257, X'8D1') Wrong version of MQMD supplied.

Usage notes for MQCB

MQCB function call - Usage notes

1. MQCB is used to define the action to be invoked for each message, matching the specified criteria, available on the queue. When the action is processed, either the message is removed from the queue and passed to the defined message consumer, or a message token is provided, which is used to retrieve the message.
2. MQCB can be used to define callback routines before starting consumption with MQCTL or it can be used from within a callback routine.
3. To use MQCB from outside of a callback routine, you must first suspend message consumption by using MQCTL and resume consumption afterwards.

Message consumer callback sequence

You can configure a consumer to invoke callback at key points during the lifecycle of the consumer. For example:

- when the consumer is first registered,
- when the connection is started,
- when the connection is stopped and
- when the consumer is deregistered, either explicitly, or implicitly by an MQCLOSE.

Table 86. MQCTL verb definitions

Verb	Meaning
MQCTL(START)	MQCTL call using the MQOP_START Operation
MQCTL(STOP)	MQCTL call using the MQOP_STOP Operation
MQCTL(WAIT)	MQCTL call using the MQOP_START_WAIT Operation

This allows the consumer to maintain state associated with the consumer. When a callback is requested by an application, the rules for consumer invocation are as follows:

REGISTER

Is always the first type of invocation of the callback.

Is always called on the same thread, as the MQCB(REGISTER) call.

START

Is always called synchronously with the MQCTL(START) verb.

- All START callbacks are completed before the MQCTL(START) verb returns.

Is on the same thread as the message delivery if THREAD_AFFINITY is requested.

The call with start is not guaranteed if, for example, a previous callback issues MQCTL(STOP) during the MQCTL(START).

STOP No further messages or events are delivered after this call until the connection is restarted.

A STOP is guaranteed if the application was previously called for START, or a message, or an event.

DEREGISTER

Is always the last type of invocation of the callback.

Ensure that your application performs thread-based initialization and cleanup in the START and STOP callbacks. You can do nonthread-based initialization and cleanup with REGISTER and DEREGISTER callbacks.

Do not make any assumptions about the life and availability of the thread other than what is stated. For example, do not rely on a thread staying alive beyond the last call to DEREGISTER. Similarly, when you have chosen not to use THREAD_AFFINITY, do not assume that the thread exists whenever the connection is started.

If your application has particular requirements for thread characteristics, it can always create a thread accordingly, then use MQCTL(WAIT). This has the effect of 'donating' the thread to MQ for asynchronous message delivery.

Message consumer connection usage

Normally, when an application issues another MQI call while one is outstanding, the call fails with reason code MQRC_CALL_IN_PROGRESS.

There are special cases, however, when the application needs to issue a further MQI call before the previous call has completed. For example, the consumer can be invoked during an MQBC call with MQOP_REGISTER.

In such an instance, when as a result of the application issuing either an MQCB or MQCTL verb, the application is called back, the application is allowed to issue a further MQI call. This means you can issue, for example, an MQOPEN call, in the consumer function when called with a CallType type of MQCBCT_REGISTER. Any MQI call, with the exception of MQDISC, is allowed.

Related reference

"Message consumer callback sequence" on page 417

Language invocations for MQCB

Manage callback function - Language invocations

The MQCB call is supported in the following programming languages.

C invocation

MQCB function call - C language invocation

```
MQCB (Hconn, Operation, CallbackDesc, Hobj, MsgDesc,  
GetMsgOpts, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;          /* Connection handle */  
MQLONG  Operation;     /* Operation being processed */  
MQCBD   CallbackDesc;  /* Callback descriptor */
```

```

|      MQHOBJ   HObj           /* Object handle */
|      MQMD     MsgDesc        /* Message descriptor attributes */
|      MQGMO    GetMsgOpts     /* Message options */
|      MQLONG   CompCode;      /* Completion code */
|      MQLONG   Reason;        /* Reason code qualifying CompCode */

```

COBOL invocation

```

|      CALL 'MQCB' USING HCONN, OPERATION, CBDESC, HOBJ, MSGDESC,
|                      GETMSGOPTS, COMPCODE, REASON.

```

Declare the parameters as follows:

```

|      ** Connection handle
|      01 HCONN   PIC S9(9) BINARY.
|      ** Operation
|      01 OPERATION PIC S9(9) BINARY.
|      ** Callback Descriptor
|      01 CBDESC.
|         COPY CMQCBDV.
|      01 HOBJ    PIC S9(9) BINARY.
|      ** Message Descriptor
|      01 MSGDESC.
|         COPY CMQMDV.
|      ** Get Message Options
|      01 GETMSGOPTS.
|         COPY CMQGMV.
|      ** Completion code
|      01 COMPCODE PIC S9(9) BINARY.
|      ** Reason code qualifying COMPCODE
|      01 REASON  PIC S9(9) BINARY.

```

PL/I invocation

```

|      call MQCB(Hconn, Operation, CallbackDesc, Hobj, MsgDesc, GetMsgOpts,
|              CompCode, Reason)

```

Declare the parameters as follows:

```

|      dc1 Hconn      fixed bin(31); /* Connection handle */
|      dc1 Operation  fixed bin(31); /* Operation */
|      dc1 CallbackDesc like MQCBDV; /* Callback Descriptor */
|      dc1 Hobj       fixed bin(31); /* Object Handle */
|      dc1 MsgDesc    like MQMDV;   /* Message Descriptor */
|      dc1 GetMsgOpts like MQGMV;   /* Get Message Options */
|      dc1 CompCode   fixed bin(31); /* Completion code */
|      dc1 Reason     fixed bin(31); /* Reason code qualifying CompCode */

```

MQCB_FUNCTION – Callback function

Callback function for CICS environments

The MQCB_FUNCTION call definition is provided solely to describe the parameters that are passed to the callback function. No entry point called MQCB_FUNCTION is actually provided by the queue manager.

The specification of the actual function to be called is an input to the MQCB call and is passed in through the MQCBD structure.

Programs invoked by CICS LINK or CICS START retrieve parameters using CICS services through named objects known as channel containers. For more information, see your CICS documentation.

The container names used are the parameter names shown in the following table:

Table 87. Parameter names used by MQCB_FUNCTION call

Name	Data type	Input/Output	Description
MsgDesc	MQMD	Input	<p>This structure describes the attributes of the message retrieved. See “MQMD – Message descriptor” on page 181 for details of this structure.</p> <p>The version of MQMD passed is the latest version of the MQMD supported.</p> <p>When a version 4 MQGMO is used to request a message handle, the MsgDesc container is passed as nulls</p> <p>This is an input to message consumers, it is not used by event handlers.</p>
GetMsgOpts	MQGMO	Input	<p>Options used to control the actions of the message consumer. It also contains additional information about the message returned. See “MQGMO – Get-message options” on page 126 for details of this structure.</p> <p>The version of MQGMO passed is the latest version of the MQGMO supported.</p> <p>This is an input to message consumers, it is not used by event handlers.</p>
Buffer	MQBYTExBufferLength	Input	<p>This is the area containing the message data.</p> <p>This is an input to message consumers, it is not used by event handlers.</p>
Context	MQCBC	Input and Output	<p>This structure provides context information to the callback functions. see “MQCBC – Callback context” on page 38 for details of this structure.</p>

Syntax for MQCB_FUNCTION

Callback function - syntax

MQCB_FUNCTION (*Hconn, MsgDesc, GetMsgOpts, Buffer, Context*)

Parameters for MQCB_FUNCTION

The MQCB_FUNCTION call has the following parameters. Callback function - parameters

Hconn (MQHCONN) – input

Callback function - Hconn parameter

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

MsgDesc (MQMD) – input

Callback function -MsgDesc parameter

This structure describes the attributes of the message retrieved.

See “MQMD – Message descriptor” on page 181 for details.

The version of MQMD passed is the same version as passed on the MQCB call that defined the consumer function.

The address of the MQMD is passed as null characters if a version 4 MQGMO was used to request that a Message Handle be returned instead of an MQMD.

This is an input field to the message consumer function; it is not relevant to an exception handler function.

GetMsgOpts (MQGMO) – input

Callback function -GetMsgOpts parameter

Options used to control the actions of the message consumer. This parameter also contains additional information about the message returned.

See MQGMO for details.

The version of MQGMO passed is the latest version supported.

This is an input field to the message consumer function; it is not relevant to an exception handler function.

Buffer (MQBYTExBufferLength) – input

Callback function - Buffer parameter

This is the area containing the message data.

If no message is available for this call, or if the message contains no message data, the address of the *Buffer* is passed as nulls.

This is an input field to the message consumer function; it is not relevant to an exception handler function.

Context – input/output

Callback function -Context parameter

This structure provides context information to the callback functions. See “MQCBC – Callback context” on page 38 for details.

Usage notes for MQCB_FUNCTION

Callback function - Usage notes

1. Be aware that if your callback routines use services that could delay or block the thread, for example, MQGET with wait, this could delay the dispatch of other callbacks.
2. A separate unit of work is not automatically established for each invocation of a callback routine, so routines can either issue a commit call, or defer committing, until a logical batch of work has been processed. When the batch

of work is committed, it commits the messages for all callback functions that have been invoked since the last syncpoint.

3. A callback routine should not, in general, rely on being invoked from the same thread each time. If this is required, use the MQCTLO_THREAD_AFFINITY when the connection is started.
4. When callback routine receives a nonzero reason code, it must take appropriate action.

Language invocations for MQCB_FUNCTION

The MQCB_FUNCTION call is supported in the programming languages shown below. Callback function - Language invocations

C invocation

MQCB_FUNCTION function call - C language invocation

```
MQCB_FUNCTION (Hconn, MsgDesc, GetMsgOpts, Buffer, &Context);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;          /* Connection handle */
MQMD     MsgDesc        /* Message descriptor attributes */
MQGMO    GetMsgOpts     /* Message options */
MQBYTE   Buffer[n];     /* Area to contain the message data */
MQCBC    Context        /* Context information */
```

COBOL invocation

```
CALL 'MQCB_FUNCTION' USING HCONN, MSGDESC, GETMSGOPTS,
                           BUFFER, CONTEXT.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN PIC S9(9) BINARY.
** Message Descriptor
01 MSGDESC.
   COPY CMQMDV.
** Get Message Options
01 GETMSGOPTS.
   COPY CMQGMV.
** Message Buffer
01 BUFFER PIC X(n)
** Consumer Context
01 CONTEXT
   COPY CMQCBCV.
```

PL/I invocation

```
call MQCB_FUNTION(Hconn, MsgDesc, GetMsgOpts, Buffer, Context)
```

Declare the parameters as follows:

```
dc1 Hconn      fixed bin(31); /* Connection handle */
dc1 MsgDesc    like MQMD;    /* Message Descriptor */
dc1 GetMsgOpts like MQGMO;   /* Get Message Options */
dc1 Buffer      pointer;      /* Pointer to message */
dc1 Context    like MQCBC;   /* Callback Context */
```

MQCLOSE – Close object

The MQCLOSE call relinquishes access to an object, and is the inverse of the MQOPEN and MQSUB calls.

Syntax for MQCLOSE

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

Parameters for MQCLOSE

The MQCLOSE call has the following parameters.

Hconn (MQHCONN) – input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on i5/OS for applications running in compatibility mode, you can omit the MQCONN call, and specify the following value for *Hconn*:

`MQHC_DEF_HCONN`

Default connection handle.

Hobj (MQHOBJ) – input/output

This handle represents the object that is being closed. The object can be of any type. The value of *Hobj* was returned by a previous MQOPEN call.

On successful completion of the call, the queue manager sets this parameter to a value that is not a valid handle for the environment. This value is:

`MQHO_UNUSABLE_HOBJ`

Unusable object handle.

On z/OS, *Hobj* is set to a value that is undefined.

Options (MQLONG) – input

This parameter controls how the object is closed.

Only permanent dynamic queues and subscriptions can be closed in more than one way, because they must be either retained or deleted; these are queues whose *DefinitionType* attribute has the value `MQQDT_PERMANENT_DYNAMIC` (see the *DefinitionType* attribute described in “Attributes for queues” on page 615). The close options are summarized in this topic.

Durable subscriptions can either be kept or removed; these are created using the MQSUB call with the `MQSO_DURABLE` option.

When closing the handle to a managed destination (that is the *Hobj* parameter returned on an MQSUB call which used the `MQSO_MANAGED` option) the queue manager will clean up any publications that have not been retrieved when the associated subscription has also been removed. The subscription is removed using the `MQCO_REMOVE_SUB` option on the *Hsub* parameter returned on an MQSUB call. Note that `MQCO_REMOVE_SUB` is the default behaviour on MQCLOSE for a non-durable subscription.

When closing a handle to a non-managed destination you are responsible for cleaning up the queue where publications are sent. Close the subscription using MQCO_REMOVE_SUB first and then process messages off the queue until there are none left.

You must specify one option only from the following:

Dynamic queue options: These options control how permanent dynamic queues are closed.

MQCO_DELETE

The queue is deleted if either of the following is true:

- It is a permanent dynamic queue, created by a previous MQOPEN call, 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, including the case where the *Hobj* was returned on an MQSUB call, the call fails with reason code MQRC_OPTION_NOT_VALID_FOR_TYPE, and the object is not deleted.

On z/OS, if the queue is a dynamic queue that has been logically deleted, and this is the last handle for it, the queue is physically deleted. See MQCLOSE usage notes for further details.

MQCO_DELETE_PURGE

The queue is deleted, and any messages on it purged, if either of the following is true:

- It is a permanent dynamic queue, created by a previous MQOPEN call, 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, including the case where the *Hobj* was returned on an MQSUB call, the call fails with reason code MQRC_OPTION_NOT_VALID_FOR_TYPE, and the object is not deleted.

The table shows which close options are valid, and whether the object is retained or deleted.

Type of object or queue	MQCO_NONE	MQCO_DELETE	MQCO_DELETE_PURGE
Object other than a queue	Retained	Not valid	Not valid
Predefined queue	Retained	Not valid	Not valid
Permanent dynamic queue	Retained	Deleted if empty and no pending updates	Messages deleted; queue deleted if no pending updates
Temporary dynamic queue (call issued by creator of queue)	Deleted	Deleted	Deleted
Temporary dynamic queue (call not issued by creator of queue)	Retained	Not valid	Not valid
Distribution list	Retained	Not valid	Not valid

The table shows which close options are valid, and whether the object is retained or deleted.

Type of object or queue	MQCO_NONE	MQCO_DELETE	MQCO_DELETE_PURGE
Managed subscription destination	Retained	Not valid	Not valid
Distribution list (subscription has been removed)	Messages deleted; queue deleted	Not valid	Not valid

Subscription closure options: These options control whether durable subscriptions are removed when the handle is closed, and whether publications still waiting to be read by the application are cleaned up. These options are only valid for use with an object handle returned in the *Hsub* parameter of an MQSUB call.

MQCO_KEEP_SUB

The handle to the subscription is closed but the subscription made is kept. Publications will continue to be sent to the destination specified in the subscription. This option is only valid if the subscription was made with the option MQSO_DURABLE.

MQCO_KEEP_SUB is the default if the subscription is durable

MQCO_REMOVE_SUB

The subscription is removed and the handle to the subscription is closed.

The *Hobj* parameter of the MQSUB call is not invalidated by closure of the *Hsub* parameter and may continue to be used for MQGET or MQCB to receive the remaining publications. When the *Hobj* parameter of the MQSUB call is also closed, if it was a managed destination any un-retrieved publications will be removed.

MQCO_REMOVE_SUB is the default if the subscription is non-durable.

These subscription closure options are summarized in the following tables.

To close a durable subscription handle but retain the subscription, use the following subscription closure options:

Task	Subscription closure option
Keep publications on an MQOPENed handle	MQCO_KEEP_SUB
Remove publications on an MQOPENed handle	Action not allowed
Keep publications on an MQSO_MANAGED handle	MQCO_KEEP_SUB
Remove publications on an MQSO_MANAGED handle	Action not allowed

To unsubscribe, either by closing a durable subscription handle and unsubscribing it or closing a non-durable subscription handle, use the following subscription closure options:

Task	Subscription closure option
Keep publications on an MQOPENed handle	MQCO_REMOVE_SUB
Remove publications on an MQOPENed handle	Action not allowed

Task	Subscription closure option
Keep publications on an MQSO_MANAGED handle	MQCO_REMOVE_SUB

Read ahead options: The following options control what happens to non-persistent messages which have been sent to the client before an application requested them and have not yet been consumed by the application. These messages are stored in the client read ahead buffer waiting to be requested by the application and can either be discarded or consumed from the queue before the MQCLOSE is completed.

MQCO_IMMEDIATE

The object is closed immediately and any messages which have been sent to the client before an application requested them are discarded and are not available to be consumed by any application. This is the default value.

MQCO QUIESCE

A request to close the object is made, but if any messages which have been sent to the client before an application requested them, still reside in the client read ahead buffer, the MQCLOSE call will return with a warning of MQRC_READ_AHEAD_MSGS and the object handle will remain valid.

The application can then continue to use the object handle to retrieve messages until no more are available, and then close the object again. No more messages will be sent to the client ahead of an application requesting them, read ahead is now turned off.

Applications are advised to use MQCO QUIESCE rather than trying to reach a point where there are no more messages in the client read ahead buffer, since a message could arrive between the last MQGET call and the following MQCLOSE which would be discarded if MQCO_IMMEDIATE was used.

If an MQCLOSE with MQCO QUIESCE is issued from within an asynchronous callback function, the same behavior of reading ahead messages applies. If the warning MQRC_READ_AHEAD_MSGS is returned, then the callback function will be called at least one more time. When the last remaining message that was read ahead has been passed to the callback function the MQCBC ConsumerFlags field is set to MQCBCF_READA_BUFFER_EMPTY.

Default option: If you require none of the options described above, you can use the following option:

MQCO_NONE

No optional close processing required.

This *must* be specified for:

- Objects other than queues
- Predefined queues
- Temporary dynamic queues (but only in those cases where *Hobj* is *not* the handle returned by the MQOPEN call that created the queue).
- Distribution lists

In all the above cases, the object is retained and not deleted.

If this option is specified for a temporary dynamic queue:

- The queue is deleted, if it was created by the MQOPEN call that returned *Hobj*; any messages that are on the queue are purged.

- In all other cases the queue (and any messages on it) are retained.

If this option is specified for a permanent dynamic queue, the queue is retained and not deleted.

On z/OS, if the queue is a dynamic queue that has been logically deleted, and this is the last handle for it, the queue is physically deleted. See MQCLOSE usage notes for further details.

CompCode (MQLONG) – output

The completion code; it is one of the following:

MQCC_OK

Successful completion.

MQCC_WARNING

Warning (partial completion).

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

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

If *CompCode* is MQCC_WARNING:

MQRC_INCOMPLETE_GROUP

(2241, X'8C1') Message group not complete.

MQRC_INCOMPLETE_MSG

(2242, X'8C2') Logical message not complete.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE

(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR

(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR

(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR

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

MQRC_ASID_MISMATCH

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

MQRC_CALL_IN_PROGRESS

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

MQRC_CF_STRUC_FAILED

(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE

(2346, X'92A') Coupling-facility structure in use.

| **MQRC_CICS_WAIT_FAILED**
 | (2140, X'85C') Wait request rejected by CICS.

| **MQRC_CONNECTION_BROKEN**
 | (2009, X'7D9') Connection to queue manager lost.

| **MQRC_CONNECTION_NOT_AUTHORIZED**
 | (2217, X'8A9') Not authorized for connection.

| **MQRC_CONNECTION_STOPPING**
 | (2203, X'89B') Connection shutting down.

| **MQRC_HCONN_ERROR**
 | (2018, X'7E2') Connection handle not valid.

| **MQRC_HOBJ_ERROR**
 | (2019, X'7E3') Object handle not valid.

| **MQRC_NOT_AUTHORIZED**
 | (2035, X'7F3') Not authorized for access.

| **MQRC_OBJECT_DAMAGED**
 | (2101, X'835') Object damaged.

| **MQRC_OPTION_NOT_VALID_FOR_TYPE**
 | (2045, X'7FD') On an MQOPEN or MQCLOSE call: option not valid for
 | object type.

| **MQRC_OPTIONS_ERROR**
 | (2046, X'7FE') Options not valid or not consistent.

| **MQRC_PAGESET_ERROR**
 | (2193, X'891') Error accessing page-set data set.

| **MQRC_Q_MGR_NAME_ERROR**
 | (2058, X'80A') Queue manager name not valid or not known.

| **MQRC_Q_MGR_NOT_AVAILABLE**
 | (2059, X'80B') Queue manager not available for connection.

| **MQRC_Q_MGR_STOPPING**
 | (2162, X'872') Queue manager shutting down.

| **MQRC_Q_NOT_EMPTY**
 | (2055, X'807') Queue contains one or more messages or uncommitted put
 | or get requests.

| **MQRC_READ_AHEAD_MSGS**
 | (nnnn, X'xxx') The client has read ahead messages that have not yet been
 | consumed by the application.

| **MQRC_RESOURCE_PROBLEM**
 | (2102, X'836') Insufficient system resources available.

| **MQRC_SECURITY_ERROR**
 | (2063, X'80F') Security error occurred.

| **MQRC_STORAGE_NOT_AVAILABLE**
 | (2071, X'817') Insufficient storage available.

| **MQRC_SUPPRESSED_BY_EXIT**
 | (2109, X'83D') Call suppressed by exit program.

| **MQRC_UNEXPECTED_ERROR**
 | (2195, X'893') Unexpected error occurred.

For detailed information on these codes, see:

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

Usage notes for MQCLOSE

Consider these points when using MQCLOSE.

1. When an application issues the MQDISC call, or ends either normally or abnormally, any objects that were opened by the application and are still open are closed automatically with the MQCO_NONE option.
2. The following points apply if the object being closed is a *queue*:
 - If operations on the queue were performed as part of a unit of work, the queue can be closed before or after the syncpoint occurs without affecting the outcome of the syncpoint. If the queue is triggered, performing a rollback before closing the queue can cause a trigger message to be issued. For details, see *WebSphere MQ Application Programming Guide*.
 - If the queue was opened with the MQOO_BROWSE option, the browse cursor is destroyed. If the queue is subsequently reopened with the MQOO_BROWSE option, a new browse cursor is created (see MQOO_BROWSE).
 - If a message is currently locked for this handle at the time of the MQCLOSE call, the lock is released (see MQGMO_LOCK).
 - On z/OS, if there is an MQGET request with the MQGMO_SET_SIGNAL option outstanding against the queue handle being closed, the request is canceled (see MQGMO_SET_SIGNAL). Signal requests for the same queue but lodged against different handles (*Hobj*) are not affected (unless a dynamic queue is being deleted, in which case they are also canceled).
3. The following points apply if the object being closed is a *dynamic queue* (either permanent or temporary):
 - For a dynamic queue, you can specify the MQCO_DELETE and MQCO_DELETE_PURGE options regardless of the options specified on the corresponding MQOPEN call.
 - When a dynamic queue is deleted, all MQGET calls with the MQGMO_WAIT option that are outstanding against the queue are canceled and reason code MQRC_Q_DELETED is returned. See MQGMO_WAIT.

Although applications cannot access a deleted queue, the queue is not removed from the system, and associated resources are not freed, until such time as all handles that reference the queue have been closed, and all units of work that affect the queue have been either committed or backed out.

On z/OS, a queue that has been logically deleted but not yet removed from the system prevents the creation of a new queue with the same name as the deleted queue; the MQOPEN call fails with reason code MQRC_NAME_IN_USE in this case. Also, such a queue can still be displayed using MQSC commands, even though it cannot be accessed by applications.
 - When a permanent dynamic queue is deleted, if the *Hobj* handle specified on the MQCLOSE call is *not* the one that was returned by the MQOPEN call that created the queue, a check is made that the user identifier that was used to validate the MQOPEN call is authorized to delete the queue. If the MQOO_ALTERNATE_USER_AUTHORITY option was specified on the MQOPEN call, the user identifier checked is the *AlternateUserId*.

This check is not performed if:

- The handle specified is the one returned by the MQOPEN call that created the queue.
- The queue being deleted is a temporary dynamic queue.

- When a temporary dynamic queue is closed, if the *Hobj* handle specified on the MQCLOSE call is the one that was returned by the MQOPEN call that created the queue, the queue is deleted. This occurs regardless of the close options specified on the MQCLOSE call. If there are messages on the queue, they are discarded; no report messages are generated.

If there are uncommitted units of work that affect the queue, the queue and its messages are still deleted, but the units of work do not fail. However, as described above, the resources associated with the units of work are not freed until each of the units of work has been either committed or backed out.

4. The following points apply if the object being closed is a *distribution list*:
 - The only valid close option for a distribution list is MQCO_NONE; the call fails with reason code MQRC_OPTIONS_ERROR or MQRC_OPTION_NOT_VALID_FOR_TYPE if any other options are specified.
 - When a distribution list is closed, individual completion codes and reason codes are not returned for the queues in the list; only the *CompCode* and *Reason* parameters of the call are available for diagnostic purposes.

If a failure occurs closing one of the queues, the queue manager continues processing and attempts to close the remaining queues in the distribution list. The *CompCode* and *Reason* parameters of the call are set to return information describing the failure. It is possible for the completion code to be MQCC_FAILED, even though most of the queues were closed successfully. The queue that encountered the error is not identified.

If there is a failure on more than one queue, it is not defined which failure is reported in the *CompCode* and *Reason* parameters.
5. On i5/OS, if the application was connected implicitly when the first MQOPEN call was issued, an implicit MQDISC occurs when the last MQCLOSE is issued.

Only applications running in compatibility mode can be connected implicitly; other applications must issue the MQCONN or MQCONNX call to connect to the queue manager explicitly.

Language invocations for MQCLOSE

The MQCLOSE call is supported in the programming languages shown below.

C invocation

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

Declare the parameters as follows:

```
MQHCONN Hconn;    /* Connection handle */
MQHOBJ  Hobj;     /* Object handle */
MQLONG  Options;  /* Options that control the action of MQCLOSE */
MQLONG  CompCode; /* Completion code */
MQLONG  Reason;   /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCLOSE' USING HCONN, HOBJ, OPTIONS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN PIC S9(9) BINARY.
** Object handle
```

```

|         01 HOBJ      PIC S9(9) BINARY.
|         ** Options that control the action of MQCLOSE
|         01 OPTIONS  PIC S9(9) BINARY.
|         ** Completion code
|         01 COMPCODE PIC S9(9) BINARY.
|         ** Reason code qualifying COMPCODE
|         01 REASON   PIC S9(9) BINARY.

```

PL/I invocation

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

Declare the parameters as follows:

```

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

```

System/390 assembler invocation

```
CALL MQCLOSE,(HCONN,HOBJ,OPTIONS,COMPCODE,REASON)
```

Declare the parameters as follows:

```

| HCONN      DS F Connection handle
| HOBJ       DS F Object handle
| OPTIONS    DS F Options that control the action of MQCLOSE
| COMPCODE   DS F Completion code
| REASON     DS F Reason code qualifying COMPCODE

```

Visual Basic invocation

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

Declare the parameters as follows:

```

| Dim Hconn    As Long 'Connection handle'
| Dim Hobj     As Long 'Object handle'
| Dim Options  As Long 'Options that control the action of MQCLOSE'
| Dim CompCode As Long 'Completion code'
| Dim Reason   As Long 'Reason code qualifying CompCode'

```

MQCMIT – Commit changes

The MQCMIT call indicates to the queue manager that the application has reached a syncpoint, and that all the message gets and puts that have occurred since the last syncpoint are to be made permanent.

Messages put as part of a unit of work are made available to other applications; messages retrieved as part of a unit of work are deleted.

- On z/OS, the call is used only by batch programs (including IMS batch DL/I programs).
- On i5/OS, this call is not supported for applications running in compatibility mode.

Syntax for MQCMIT

```
MQCMIT (Hconn, CompCode, Reason)
```

Parameters for MQCMIT

The MQCMIT call has the following parameters.

Hconn (MQHCONN) – input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNEX call.

CompCode (MQLONG) – output

The completion code; it is one of the following:

MQCC_OK

Successful completion.

MQCC_WARNING

Warning (partial completion).

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

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

If *CompCode* is MQCC_WARNING:

MQRC_BACKED_OUT

(2003, X'7D3') Unit of work backed out.

MQRC_OUTCOME_PENDING

(2124, X'84C') Result of commit operation is pending.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_SERV_LOAD_ERROR

(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR

(2374, X'946') API exit failed.

MQRC_ASID_MISMATCH

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

MQRC_CALL_IN_PROGRESS

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

MQRC_CF_STRUC_IN_USE

(2346, X'92A') Coupling-facility structure in use.

MQRC_CONNECTION_BROKEN

(2009, X'7D9') Connection to queue manager lost.

MQRC_ENVIRONMENT_ERROR

(2012, X'7DC') Call not valid in environment.

MQRC_HCONN_ERROR

(2018, X'7E2') Connection handle not valid.

MQRC_OBJECT_DAMAGED

(2101, X'835') Object damaged.

MQRC_OUTCOME_MIXED

(2123, X'84B') Result of commit or back-out operation is mixed.

MQRC_Q_MGR_STOPPING

(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM

(2102, X'836') Insufficient system resources available.

MQRC_STORAGE_MEDIUM_FULL

(2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE

(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR

(2195, X'893') Unexpected error occurred.

For detailed information on these codes, see:

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

Usage notes for MQCMIT

1. Use this call only when the queue manager itself coordinates the unit of work. This can be:

- A local unit of work, where the changes affect only MQ resources.
- A global unit of work, where the changes can affect resources belonging to other resource managers, as well as affecting MQ resources.

For further details about local and global units of work, see “MQBEGIN – Begin unit of work” on page 401.

2. In environments where the queue manager does not coordinate the unit of work, the appropriate commit call must be used instead of MQCMIT. The environment might also support an implicit commit caused by the application terminating normally.
 - On z/OS, use the following calls:
 - Batch programs (including IMS batch DL/I programs) can use the MQCMIT call if the unit of work affects only MQ resources. However, if the unit of work affects both MQ resources and resources belonging to other resource managers (for example, DB2), use the SRRRCMIT call provided by the z/OS Recoverable Resource Service (RRS). The SRRRCMIT call commits changes to resources belonging to the resource managers that have been enabled for RRS coordination.
 - CICS applications must use the EXEC CICS SYNCPOINT command to commit the unit of work explicitly. Alternatively, ending the transaction results in an implicit commit of the unit of work. The MQCMIT call cannot be used for CICS applications.
 - IMS applications (other than batch DL/I programs) must use IMS calls such as GU and CHKP to commit the unit of work. The MQCMIT call cannot be used for IMS applications (other than batch DL/I programs).
 - On i5/OS, use this call for local units of work coordinated by the queue manager. This means that a commitment definition must not exist at job level, that is, the STRCMTCTL command with the CMTSCOPE(*JOB) parameter must not have been issued for the job.

3. If an application ends with uncommitted changes in a unit of work, the disposition of those changes depends on whether the application ends normally or abnormally. See MQDISC usage notes for further details.
4. When an application puts or gets messages in groups or segments of logical messages, the queue manager retains information relating to the message group and logical message for the last successful MQPUT and MQGET calls. This information is associated with the queue handle, and includes such things as:
 - The values of the *GroupId*, *MsgSeqNumber*, *Offset*, and *MsgFlags* fields in MQMD.
 - Whether the message is part of a unit of work.
 - For the MQPUT call: whether the message is persistent or nonpersistent.

When a unit of work is committed, the queue manager retains the group and segment information, and the application can continue putting or getting messages in the current message group or logical message.

Retaining the group and segment information when a unit of work is committed allows the application to spread a large message group or large logical message consisting of many segments across several units of work. Using several units of work is advantageous if the local queue manager has only limited queue storage. However, the application must maintain sufficient information to restart putting or getting messages at the correct point in the event that a system failure occurs. For details of how to restart at the correct point after a system failure, see MQPMO_LOGICAL_ORDER and MQGMO_LOGICAL_ORDER.

The remaining usage notes apply only when the queue manager coordinates the units of work:

5. A unit of work has the same scope as a connection handle; all MQ calls that affect a particular unit of work must be performed using the same connection handle. Calls issued using a different connection handle (for example, calls issued by another application) affect a different unit of work. See the *Hconn* parameter described in MQCONN for information about the scope of connection handles.
6. Only messages that were put or retrieved as part of the current unit of work are affected by this call.
7. A long-running application that issues MQGET, MQPUT, or MQPUT1 calls within a unit of work, but that never issues a commit or back-out call, can fill queues with messages that are not available to other applications. To guard against this, the administrator must set the *MaxUncommittedMsgs* queue-manager attribute to a value that is low enough to prevent runaway applications filling the queues, but high enough to allow the expected messaging applications to work correctly.
8. On UNIX and Windows systems, if the *Reason* parameter is MQRC_CONNECTION_BROKEN (with a *CompCode* of MQCC_FAILED), or MQRC_UNEXPECTED_ERROR it is possible that the unit of work was successfully committed.

Language invocations for MQCMIT

The MQCMIT call is supported in the programming languages shown below.

C invocation

```
MQCMIT (Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```

MQHCONN Hconn;    /* Connection handle */
MQLONG  CompCode; /* Completion code */
MQLONG  Reason;   /* Reason code qualifying CompCode */

```

COBOL invocation

```
CALL 'MQCMIT' USING HCONN, COMPCODE, REASON.
```

Declare the parameters as follows:

```

** Connection handle
01 HCONN    PIC S9(9) BINARY.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON   PIC S9(9) BINARY.

```

PL/I invocation

```
call MQCMIT (Hconn, CompCode, Reason);
```

Declare the parameters as follows:

```

dcl Hconn    fixed bin(31); /* Connection handle */
dcl CompCode fixed bin(31); /* Completion code */
dcl Reason   fixed bin(31); /* Reason code qualifying CompCode */

```

System/390 assembler invocation

```
CALL MQCMIT,(HCONN,COMPCODE,REASON)
```

Declare the parameters as follows:

```

HCONN      DS F Connection handle
COMPCODE   DS F Completion code
REASON     DS F Reason code qualifying COMPCODE

```

Visual Basic invocation

```
MQCMIT Hconn, CompCode, Reason
```

Declare the parameters as follows:

```

Dim Hconn    As Long 'Connection handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'

```

MQCONN – Connect queue manager

The MQCONN call connects an application program to a queue manager.

It provides a queue manager connection handle, which the application uses on subsequent message queuing calls.

- On z/OS, CICS applications do not have to issue this call. These applications are connected automatically to the queue manager to which the CICS system is connected. However, the MQCONN and MQDISC calls are still accepted from CICS applications.
- On i5/OS, applications running in compatibility mode do not have to issue this call. These applications are connected automatically to the queue manager when they issue the first MQOPEN call. However, the MQCONN and MQDISC calls are still accepted from i5/OS applications.

Other applications (that is, applications not running in compatibility mode) must use the MQCONN or MQCONNX call to connect to the queue manager, and the MQDISC call to disconnect from the queue manager. This is the recommended style of programming.

Syntax for MQCONN

MQCONN (*QMgrName*, *Hconn*, *CompCode*, *Reason*)

Parameters for MQCONN

The MQCONN call has the following parameters.

QMgrName (MQCHAR48) – input

Queue manager names can contain certain characters, and there are other restrictions on their formation. You can use special values of QMgrName to indicate a default queue manager, a queue-sharing group, or a choice of queue managers.

This is the name of the queue manager to which the application wants to connect. The name can contain the following characters:

- Uppercase alphabetic characters (A through Z)
- Lowercase alphabetic characters (a through z)
- Numeric digits (0 through 9)
- Period (.), forward slash (/), underscore (_), percent (%)

The name must not contain leading or embedded blanks, but can contain trailing blanks. A null character can be used to indicate the end of significant data in the name; the null and any characters following it are treated as blanks. The following restrictions apply in the environments indicated:

- On systems that use EBCDIC Katakana, lowercase characters cannot be used.
- On z/OS, names that begin or end with an underscore cannot be processed by the operations and control panels. For this reason, avoid such names.
- On i5/OS, enclose names containing lowercase characters, forward slash, or percent in quotation marks when specified on commands. Do not specify these quotation marks in the *QMgrName* parameter.

If the name consists entirely of blanks, the name of the *default* queue manager is used.

The name specified for *QMgrName* must be the name of a *connectable* queue manager.

On z/OS, the queue managers to which it is possible to connect are determined by the environment:

- For CICS, you can use only the queue manager to which the CICS system is connected. The *QMgrName* parameter must still be specified, but its value is ignored; blanks are recommended.
- For IMS, only queue managers that are listed in the subsystem definition table (CSQQDEFV), and listed in the SSM table in IMS, are connectable (see usage note 6 on page 442).
- For z/OS batch and TSO, only queue managers that reside on the same system as the application are connectable (see usage note 6 on page 442).

Queue-sharing groups: On systems where several queue managers exist and are configured to form a queue-sharing group, the name of the queue-sharing group can be specified for *QMgrName* in place of the name of a queue manager. This allows the application to connect to *any* queue manager that is available in the

queue-sharing group and that is on the same z/OS image as the application. The system can also be configured so that using a blank *QMgrName* connects to the queue-sharing group instead of to the default queue manager.

If *QMgrName* specifies the name of the queue-sharing group, but there is also a queue manager with that name on the system, connection is made to the latter in preference to the former. Only if that connection fails is connection to one of the queue managers in the queue-sharing group attempted.

If the connection is successful, you can use the handle returned by the MQCONN or MQCONNX call to access *all* the resources (both shared and nonshared) that belong to the queue manager to which connection has been made. Access to these resources is subject to the usual authorization controls.

If the application issues two MQCONN or MQCONNX calls to establish concurrent connections, and one or both calls specifies the name of the queue-sharing group, the second call returns completion code MQCC_WARNING and reason code MQRC_ALREADY_CONNECTED when it connects to the same queue manager as the first call.

Queue-sharing groups are supported only on z/OS. Connection to a queue-sharing group is supported only in the batch, RRS batch, and TSO environments.

MQ client applications: For MQ client applications, a connection is attempted for each client-connection channel definition with the specified queue-manager name, until one is successful. The queue manager, however, must have the same name as the specified name. If an all-blank name is specified, each client-connection channel with an all-blank queue-manager name is tried until one is successful; in this case there is no check against the actual name of the queue manager.

MQ client applications are not supported in z/OS, but z/OS can act as an MQ server, to which MQ client applications can connect.

MQ client queue-manager groups: If the specified name starts with an asterisk (*), the queue manager to which connection is made might have a different name 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 until one is found to which a connection can be made. The order in which connections are attempted is influenced by the client channel weight and connection affinity values of the candidate channels. If none of the queue managers in the group is available for connection, the call fails. Each queue manager is tried once only. If an asterisk alone is specified for the name, an implementation-defined default queue-manager group is used.

Queue-manager groups are supported only for applications running in an MQ-client environment; the call fails if a non-client application specifies a queue-manager name beginning with an asterisk. A group is defined by providing several client connection channel definitions with the same queue-manager name (the specified name without the asterisk), to communicate with each of the queue managers in the group. The default group is defined by providing one or more client connection channel definitions, each with a blank queue-manager name (specifying an all-blank name therefore has the same effect as specifying a single asterisk for the name for a client application).

After connecting to one queue manager of a group, an application can specify blanks in the usual way in the queue-manager name fields in the message and object descriptors to mean the name of the queue manager to which the application has connected (the *local queue manager*). If the application needs to know this name, use the MQINQ call to inquire the *QMgrName* queue-manager attribute.

Prefixing an asterisk to the connection name implies that the application does not depend on connecting to a particular queue manager in the group. Suitable applications are:

- Applications that put messages but do not get messages.
- Applications that put request messages and then get the reply messages from a *temporary dynamic* queue.

Unsuitable applications are those that need to get messages from a particular queue at a particular queue manager; such applications must not prefix the name with an asterisk.

If you specify an asterisk, the maximum length of the remainder of the name is 47 characters.

Queue-manager groups are not supported on z/OS.

The length of this parameter is given by MQ_Q_MGR_NAME_LENGTH.

Hconn (MQHCONN) – output

This handle represents the connection to the queue manager. Specify it on all subsequent message queuing calls issued by the application. It ceases to be valid when the MQDISC call is issued, or when the unit of processing that defines the scope of the handle terminates.

Handle scope: The scope of the handle returned depends on the call used to connect to the queue manager (MQCONN or MQCONNX). If the call used is MQCONNX, the scope of the handle also depends on the MQCNO_HANDLE_SHARE_* option specified in the *Options* field of the MQCNO structure.

- If the call is MQCONN, or the MQCNO_HANDLE_SHARE_NONE option is specified, the handle returned is a *nonshared* handle.
The scope of a nonshared handle is the smallest unit of parallel processing supported by the platform on which the application is running (see Table 88 on page 439 for details); the handle is not valid outside the unit of parallel processing from which the call was issued.
- If you specify the MQCNO_HANDLE_SHARE_BLOCK or MQCNO_HANDLE_SHARE_NO_BLOCK option, the handle returned is a *shared* handle.

The scope of a shared handle is the process that owns the thread from which the call was issued; the handle can be used from any thread that belongs to that process. Not all platforms support threads.

The scope of nonshared handles on various platforms is shown in Table 88 on page 439.

Table 88. Scope of nonshared handles on various platforms

Platform	Scope of nonshared handle
z/OS	<ul style="list-style-type: none"> • CICS: the CICS task • IMS: the task, up to the next syncpoint (excluding subtasks of the task) • z/OS batch and TSO: the task (excluding subtasks of the task)
i5/OS	Job
UNIX systems	Thread
16-bit Windows applications	Process
32-bit Windows applications	Thread

On z/OS for CICS applications, and on i5/OS for applications running in compatibility mode, the value returned is:

MQHC_DEF_HCONN
Default connection handle.

CompCode (MQLONG) – output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason (MQLONG) – output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

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

If *CompCode* is MQCC_WARNING:

MQRC_ALREADY_CONNECTED
(2002, X'7D2') Application already connected.

MQRC_CLUSTER_EXIT_LOAD_ERROR
(2267, X'8DB') Unable to load cluster workload exit.

MQRC_SSL_ALREADY_INITIALIZED
(2391, X'957') SSL already initialized.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_CONN_LOAD_ERROR
(2129, X'851') Unable to load adapter connection module.

MQRC_ADAPTER_DEFS_ERROR
(2131, X'853') Adapter subsystem definition module not valid.

MQRC_ADAPTER_DEFS_LOAD_ERROR
(2132, X'854') Unable to load adapter subsystem definition module.

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ADAPTER_STORAGE_SHORTAGE
(2127, X'84F') Insufficient storage for adapter.

MQRC_ANOTHER_Q_MGR_CONNECTED
(2103, X'837') Another queue manager already connected.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_INIT_ERROR
(2375, X'947') API exit initialization failed.

MQRC_API_EXIT_TERM_ERROR
(2376, X'948') API exit termination failed.

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

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

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

MQRC_CONN_ID_IN_USE
(2160, X'870') Connection identifier already in use.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_ERROR
(2273, X'8E1') Error processing MQCONN call.

MQRC_CONNECTION QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_CRYPTO_HARDWARE_ERROR
(2382, X'94E') Cryptographic hardware configuration error.

MQRC_DUPLICATE_RECOV_COORD
(2163, X'873') Recovery coordinator already exists.

MQRC_ENVIRONMENT_ERROR
(2012, X'7DC') Call not valid in environment.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOST_NOT_AVAILABLE
(2538, X'9EA') The attempt to allocate a conversation to the remote system failed.

MQRC_KEY_REPOSITORY_ERROR
(2381, X'94D') Key repository not valid.

- MQRC_MAX_CONNS_LIMIT_REACHED**
(2025, X'7E9') Maximum number of connections reached.
- MQRC_NOT_AUTHORIZED**
(2035, X'7F3') Not authorized for access.
- MQRC_OPEN_FAILED**
(2137, X'859') Object not opened successfully.
- MQRC_Q_MGR_NAME_ERROR**
(2058, X'80A') Queue manager name not valid or not known.
- MQRC_Q_MGR_NOT_AVAILABLE**
(2059, X'80B') Queue manager not available for connection.
- MQRC_Q_MGR QUIESCING**
(2161, X'871') Queue manager quiescing.
- MQRC_Q_MGR_STOPPING**
(2162, X'872') Queue manager shutting down.
- MQRC_RESOURCE_PROBLEM**
(2102, X'836') Insufficient system resources available.
- MQRC_SECURITY_ERROR**
(2063, X'80F') Security error occurred.
- MQRC_SSL_INITIALIZATION_ERROR**
(2393, X'959') SSL initialization error.
- MQRC_STORAGE_NOT_AVAILABLE**
(2071, X'817') Insufficient storage available.
- MQRC_UNEXPECTED_ERROR**
(2195, X'893') Unexpected error occurred.

For detailed information on these codes, see:

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

Usage notes for MQCONN

1. The queue manager to which connection is made using the MQCONN call is called the *local queue manager*.
2. Queues that are owned by the local queue manager appear to the application as local queues. It is possible to put messages on and get messages from these queues.
Shared queues that are owned by the queue-sharing group to which the local queue manager belongs appear to the application as local queues. It is possible to put messages on and get messages from these queues.
Queues that are owned by remote queue managers appear as remote queues. It is possible to put messages on these queues, but not to get messages from these queues.
3. If the queue manager fails while an application is running, the application must issue the MQCONN call again to obtain a new connection handle to use on subsequent MQ calls. The application can issue the MQCONN call periodically until the call succeeds.

If an application is not sure whether it is connected to the queue manager, the application can safely issue an MQCONN call to obtain a connection handle. If the application is already connected, the handle returned is the same as that

returned by the previous MQCONN call, but with completion code MQCC_WARNING and reason code MQRC_ALREADY_CONNECTED.

4. When the application has finished using MQ calls, the application must use the MQDISC call to disconnect from the queue manager.

5. On z/OS:

- Batch, TSO, and IMS applications must issue the MQCONN call to use the other MQ calls. These applications can connect to more than one queue manager concurrently.

If the queue manager fails, the application must issue the call again after the queue manager has restarted to obtain a new connection handle.

Although IMS applications can issue the MQCONN call repeatedly, even when already connected, this is not recommended for online message processing programs (MPPs).

- CICS applications do not have to issue the MQCONN call to use the other MQ calls, but can do so if they want; both the MQCONN call and the MQDISC call are accepted. However, it is not possible to connect to more than one queue manager concurrently.

If the queue manager fails, these applications are automatically reconnected when the queue manager restarts, and so do not need to issue the MQCONN call.

6. On z/OS, to define the available queue managers:

- For batch applications, system programmers can use the CSQBDEF macro to create a module (CSQBDEFV) that defines the default queue-manager name, or queue-sharing group name.
- For IMS applications, system programmers can use the CSQQDEFX macro to create a module (CSQQDEFV) that defines the names of the available queue managers and specifies the default queue manager.

In addition, each queue manager must be defined to the IMS control region and to each dependent region accessing that queue manager. To do this, you must create a subsystem member in the IMS.PROCLIB library and identify the subsystem member to the applicable IMS regions. If an application attempts to connect to a queue manager that is not defined in the subsystem member for its IMS region, the application abends.

For more information on using these macros, see the *WebSphere MQ for z/OS System Setup Guide*.

7. On i5/OS, applications written for previous releases of the queue manager can run without recompiling. This is called *compatibility mode*. This mode of operation provides a compatible run-time environment for applications. It comprises the following:

- The service program AMQZSTUB residing in the library QMQM.

AMQZSTUB provides the same public interface as previous releases, and has the same signature. Use this service program to access the MQI through bound procedure calls.

- The program QMQM residing in the library QMQM.

QMQM provides a means of accessing the MQI through dynamic program calls.

- Programs MQCLOSE, MQCONN, MQDISC, MQGET, MQINQ, MQOPEN, MQPUT, MQPUT1, and MQSET residing in the library QMQM.

These programs also provide a means of accessing the MQI through dynamic program calls, but with a parameter list that corresponds to the standard descriptions of the MQ calls.

These three interfaces do not include capabilities that were introduced in MQSeries Version 5.1. For example, the MQBACK, MQCMIT, and MQCONNX calls are not supported. The support provided by these interfaces is for single-threaded applications only.

Support for the new MQ calls in single-threaded applications, and for all MQ calls in multi-threaded applications, is provided through the service programs LIBMQM and LIBMQM_R.

8. On i5/OS, programs that end abnormally are not automatically disconnected from the queue manager. Write applications to allow for the possibility of the MQCONN or MQCONNX call returning completion code MQCC_WARNING and reason code MQRC_ALREADY_CONNECTED. Use the connection handle returned in this situation as normal.

Language invocations for MQCONN

The MQCONN call is supported in the programming languages shown below.

C invocation

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

Declare the parameters as follows:

```
MQCHAR48 QMgrName; /* Name of queue manager */
MQHCONN  Hconn;    /* Connection handle */
MQLONG   CompCode; /* Completion code */
MQLONG   Reason;   /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCONN' USING QMGRNAME, HCONN, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Name of queue manager
01 QMGRNAME PIC X(48).
** Connection handle
01 HCONN PIC S9(9) BINARY.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON PIC S9(9) BINARY.
```

PL/I invocation

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

Declare the parameters as follows:

```
dc1 QMgrName char(48); /* Name of queue manager */
dc1 Hconn fixed bin(31); /* Connection handle */
dc1 CompCode fixed bin(31); /* Completion code */
dc1 Reason fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
CALL MQCONN,(QMGRNAME,HCONN,COMPCODE,REASON)
```

Declare the parameters as follows:

```
QMGRNAME DS CL48 Name of queue manager
HCONN DS F Connection handle
COMPCODE DS F Completion code
REASON DS F Reason code qualifying COMPCODE
```


Visual Basic invocation

MQCONN QMgrName, Hconn, CompCode, Reason

Declare the parameters as follows:

```
Dim QMgrName As String*48 'Name of queue manager'  
Dim Hconn As Long 'Connection handle'  
Dim CompCode As Long 'Completion code'  
Dim Reason As Long 'Reason code qualifying CompCode'
```

MQCONNX – Connect queue manager (extended)

The MQCONNX call connects an application program to a queue manager. It provides a queue manager connection handle, which is used by the application on subsequent MQ calls.

The MQCONNX call is similar to the MQCONN call, except that MQCONNX allows options to be specified to control the way that the call works.

- This call is supported on all WebSphere MQ systems, and WebSphere MQ clients connected to these systems.
- On i5/OS, this call is not supported for applications running in compatibility mode.

Syntax for MQCONNX

MQCONNX (*QMgrName*, *ConnectOpts*, *Hconn*, *CompCode*, *Reason*)

Parameters for MQCONNX

The MQCONNX call has the following parameters.

QMgrName (MQCHAR48) – input

See the *QMgrName* parameter described in “MQCONN – Connect queue manager” on page 435 for details.

ConnectOpts (MQCNO) – input/output

See “MQCNO – Connect options” on page 77 for details.

Hconn (MQHCONN) – output

See the *Hconn* parameter described in “MQCONN – Connect queue manager” on page 435 for details. This parameter forms one part of a unique identifier that allows WebSphere MQ to reliably identify an application if you disconnect it from the queue manager. It is given a 24 byte connection identifier and the value of *Hconn* is formed by taking the last eight bytes of the identifier and converting it to its 16 character hexadecimal equivalent.

Note: You can specify *Hconn* only, for connections associated with queue managers that MQSC is being run against.

CompCode (MQLONG) – output

See the *CompCode* parameter described in “MQCONN – Connect queue manager” on page 435 for details.

Reason (MQLONG) – output

The reason code qualifying *CompCode*

See the *Reason* parameter for MQCONN, described in “Reason (MQLONG) – output” on page 439 for details of possible reason codes.

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

If *CompCode* is MQCC_FAILED:

MQRC_AIR_ERROR

(2385, X'951') Authentication information record not valid.

MQRC_AUTH_INFO_CONN_NAME_ERROR

(2387, X'953') Authentication information connection name not valid.

MQRC_AUTH_INFO_REC_COUNT_ERROR

(2383, X'94F') Authentication information record count not valid.

MQRC_AUTH_INFO_REC_ERROR

(2384, X'950') Authentication information record fields not valid.

MQRC_AUTH_INFO_TYPE_ERROR

(2386, X'952') Authentication information type not valid.

MQRC_CD_ERROR

(2277, X'8E5') Channel definition not valid.

MQRC_CLIENT_CONN_ERROR

(2278, X'8E6') Client connection fields not valid.

MQRC_CNO_ERROR

(2139, X'85B') Connect-options structure not valid.

MQRC_CONN_TAG_IN_USE

(2271, X'8DF') Connection tag in use.

MQRC_CONN_TAG_NOT_USABLE

(2350, X'92E') Connection tag not usable.

MQRC_LDAP_PASSWORD_ERROR

(2390, X'956') LDAP password not valid.

MQRC_LDAP_USER_NAME_ERROR

(2388, X'954') LDAP user name fields not valid.

MQRC_LDAP_USER_NAME_LENGTH_ERR

(2389, X'955') LDAP user name length not valid.

MQRC_OPTIONS_ERROR

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

MQRC_SCO_ERROR

(2380, X'94C') SSL configuration options structure not valid.

MQRC_SSL_CONFIG_ERROR

(2392, X'958') SSL configuration error.

For detailed information on these codes, see:

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

Usage notes for MQCONNX

For the Visual Basic programming language, the following point applies:

- The *ConnectOpts* parameter is declared as being of type MQCNO. If the application is running as a WebSphere MQ client, and you want to specify the parameters of the client-connection channel, use the MQCONNXAny call in place of MQCONNX, so that the application can specify an MQCNOCD structure on the call in place of an MQCNO structure.

The MQCONNXAny call has the same parameters as the MQCONNX call, except that the *ConnectOpts* parameter is declared as being of type Any, allowing either an MQCNO structure or an MQCNOCD structure to be specified as the parameter. However, this means that the *ConnectOpts* parameter cannot be checked to ensure that it is the correct data type.

Language invocations for MQCONNX

The MQCONNX call is supported in the programming languages shown below.

C invocation

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

Declare the parameters as follows:

```
MQCHAR48 QMgrName;    /* Name of queue manager */
MQCNO    ConnectOpts; /* Options that control the action of MQCONNX */
MQHCONN  Hconn;       /* Connection handle */
MQLONG   CompCode;    /* Completion code */
MQLONG   Reason;      /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCONNX' USING QMGRNAME, CONNECTOPTS, HCONN, COMPCODE,
                    REASON.
```

Declare the parameters as follows:

```
** Name of queue manager
01 QMGRNAME    PIC X(48).
** Options that control the action of MQCONNX
01 CONNECTOPTS.
   COPY CMQCNOV.
** Connection handle
01 HCONN       PIC S9(9) BINARY.
** Completion code
01 COMPCODE    PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON      PIC S9(9) BINARY.
```

PL/I invocation

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

Declare the parameters as follows:

```
dc1 QMgrName    char(48);    /* Name of queue manager */
dc1 ConnectOpts like MQCNO;  /* Options that control the action of
                             MQCONNX */
dc1 Hconn       fixed bin(31); /* Connection handle */
dc1 CompCode    fixed bin(31); /* Completion code */
dc1 Reason      fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
CALL MQCONN, (QMGRNAME,CONNECTOPTS,HCONN,COMP CODE,REASON)
```

Declare the parameters as follows:

QMGRNAME	DS	CL48	Name of queue manager
CONNECTOPTS	CMQCNOA	,	Options that control the action of MQCONN
HCONN	DS	F	Connection handle
COMP CODE	DS	F	Completion code
REASON	DS	F	Reason code qualifying COMP CODE

Visual Basic invocation

```
MQCONN QMgrName, ConnectOpts, Hconn, CompCode, Reason
```

Declare the parameters as follows:

```
Dim QMgrName As String*48 'Name of queue manager'  
Dim ConnectOpts As MQCNO 'Options that control the action of'  
                        'MQCONN'  
Dim Hconn As Long 'Connection handle'  
Dim CompCode As Long 'Completion code'  
Dim Reason As Long 'Reason code qualifying CompCode'
```

MQCRTMH – Create message handle

The MQCRTMH call returns a message handle. An application can use it on subsequent message queuing calls:

- Use the MQSETMP call to set a property of the message handle.
- Use the MQINQMP call to inquire on the value of a property of the message handle.
- Use the MQDLTMP call to delete a property of the message handle.

The message handle can be used on the MQPUT and MQPUT1 calls to associate the properties of the message handle with those of the message being put. Similarly by specifying a message handle on the MQGET call, the properties of the message being retrieved can be accessed using the message handle when the MQGET call completes.

Use MQDLTMH to delete the message handle.

Syntax for MQCRTMH

```
MQCRTMH (Hconn, CrtMsgHOpts, Hmsg, CompCode, Reason)
```

Parameters for MQCRTMH

The MQCRTMH call has the following parameters.

Hconn (MQHCONN) – input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call. If the connection to the queue manager ceases to be valid and no WebSphere MQ call is operating on the message handle, MQDLTMH is implicitly called to delete the message.

Alternatively, you can specify the following value:

MQHC_UNASSOCIATED_HCONN

The connection handle does not represent a connection to any particular queue manager.

When this value is used, the message handle must be deleted with an explicit call to MQDLTMH in order to release any storage allocated to it; WebSphere MQ never implicitly deletes the message handle.

There must be at least one valid connection to a queue manager established on the thread creating the message handle, otherwise the call fails with MQRC_HCONN_ERROR.

On z/OS for CICS applications, and on i5/OS for applications running in compatibility mode, the MQCONN call can be omitted, and you can specify the following value for *Hconn*:

MQHC_DEF_CONN

Default connection handle

CrtMsgHOpts (MQCMHO) – input

The options that control the action of MQCRTMH. See MQCMHO for details.

Hmsg (MQHMSG) – output

On output a message handle is returned that can be used to set, inquire and delete properties of the message handle. Initially the message handle contains no properties.

A message handle also has an associated message descriptor. Initially this contains the default values. The values of the associated message descriptor fields can be set and inquired using the MQSETMP and MQINQMP calls. The MQDLTMP call will reset a field of the message descriptor back to its default value.

If the *Hconn* parameter is specified as the value MQHC_UNASSOCIATED_HCONN then the returned message handle can be used on MQGET, MQPUT, or MQPUT1 calls with any connection within the unit of processing, but can only be in use by one WebSphere MQ call at a time. If the handle is in use when a second WebSphere MQ call attempts to use the same message handle, the second WebSphere MQ call fails with reason code MQRC_MSG_HANDLE_IN_USE.

If the *Hconn* parameter is not MQHC_UNASSOCIATED_HCONN then the returned message handle can only be used on the specified connection.

The same *Hconn* parameter value must be used on the subsequent MQI calls where this message handle is used:

- MQDLTMH
- MQSETMP
- MQINQMP
- MQDLTMP
- MQMHBUF
- MQBUFMH

The returned message handle ceases to be valid when the MQDLTMH call is issued for the message handle, or when the unit of processing that defines the

scope of the handle terminates. MQDLTMH is called implicitly if a specific connection is supplied when the message handle is created and the connection to the queue manager ceases to be valid, for example, if MQDBC is called..

CompCode (MQLONG) – output

The completion code; it is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

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

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE

(2204, X'089C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR

(2130, X'852') Unable to load adapter service module.

MQRC_ASID_MISMATCH

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

MQRC_CALL_IN_PROGRESS

(2219, X'08AB') MQI call entered before previous call completed.

MQRC_CMHO_ERROR

(2461, X'099D') Create message handle options structure not valid.

MQRC_CONNECTION_BROKEN

(2273, X'7D9') Connection to queue manager lost.

MQRC_HANDLE_NOT_AVAILABLE

(2017, X'07E1') No more handles available.

MQRC_HCONN_ERROR

(2018, X'7E2') Connection handle not valid.

MQRC_HMSG_ERROR

(2460, X'099C') Message handle pointer not valid.

MQRC_OPTIONS_ERROR

(2046, X'07FE') Options not valid or not consistent.

MQRC_STORAGE_NOT_AVAILABLE

(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR

(2195, X'893') Unexpected error occurred.

See Chapter 4, "Return codes," on page 661 for more details.

Language invocations for MQCRTMH

The MQCRTMH call is supported in the programming languages shown below.

C invocation

```
MQCRTMH (Hconn, &CrtMsgH0pts, &Hmsg, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;      /* Connection handle */
MQCMHO   CrtMsgH0pts; /* Options that control the action of MQCRTMH */
MQHMSG   Hmsg;       /* Message handle */
MQLONG   CompCode;   /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCRTMH' USING HCONN, CRTMSGOPTS, HMSG, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN    PIC S9(9) BINARY.
** Options that control the action of MQCRTMH
01 CRTMSGHOPTS.
   COPY CMQCMHOV.
** Message handle
01 HMSG     PIC S9(18) BINARY.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON   PIC S9(9) BINARY.
```

PL/I invocation

```
call MQCRTMH (Hconn, CrtMsgH0pts, Hmsg, CompCode, Reason);
```

Declare the parameters as follows:

```
dc1 Hconn      fixed bin(31); /* Connection handle */
dc1 CrtMsgH0pts like MQCMHO; /* Options that control the action of MQCRTMH */
dc1 Hmsg       fixed bin(63); /* Message handle */
dc1 CompCode   fixed bin(31); /* Completion code */
dc1 Reason     fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
CALL MQCRTMH,(HCONN,CRTMSGHOPTS,HMSG,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN      DS      F Connection handle
CRTMSGHOPTS CMQCMHOA , Options that control the action of MQCRTMH
HMSG       DS      D Message handle
COMPCODE   DS      F Completion code
REASON     DS      F Reason code qualifying COMPCODE
```

MQCTL – Control callback

The MQCTL call performs controlling actions on the object handles opened for a connection.

Syntax for MQCTL

Control callback function - syntax

MQCTL (*Hconn*, *Operation*, *ControlOpts*, *CompCode*, *Reason*)

Parameters for MQCTL

The MQCTL call has the following parameters. Control callback function - parameters

Hconn (MQHCONN) – input

Control callback function - Hconn parameter

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on i5/OS for applications running in compatibility mode, the MQCONN call can be omitted, and you can specify the following special value for *Hconn*:

MQHC_DEF_HCONN

Default connection handle.

Operation (MQLONG) – input

Control callback function - Operation parameter

The operation being processed on the callback defined for the specified object handle. You must specify one, and one only, of the following options:

MQOP_START

Start the consuming of messages for all defined message consumer functions for the specified connection handle.

Callbacks run on a thread started by the system, which is different from any of the application threads.

This operation gives control of the provided connection handle to system. The only MQI calls which can be issued by a thread other than the consumer thread are:

- MQCTL with Operation MQOP_STOP
- MQCTL with Operation MQOP_SUSPEND
- MQDISC - This performs MQCTL with Operation MQOP_STOP before disconnection the HConn.

MQRC_HCONN_ASYNC_ACTIVE is returned if a WebSphere MQ API call is issued while the connection handle is started, and the call does not originate from a message consumer function.

If a message consumer stops the connection during the MQCBCT_START_CALL then the MQCTL call will return with a failure reason code of MQRC_CONNECTION_STOPPED.

This can be issued in a consumer function. For the same connection as the callback routine, its only purpose is to cancel a previously issued MQOP_STOP operation.

This option is not supported in the following environments: CICS on z/OS or if the application is bound with a nonthreaded WebSphere MQ library.

MQOP_START_WAIT

Start the consuming of messages for all defined message consumer functions for the specified connection handle.

Message consumers run on the same thread and control is not returned to the caller of MQCTL until:

- Released by the use of the MQCTL MQOP_STOP or MQOP_SUSPEND operations, or
- All consumer routines have been deregistered or suspended.

If all consumers are deregistered or suspended, an implicit MQOP_STOP operation is issued.

This option cannot be used from within a callback routine, either for the current connection handle or any other connection handle. If the call is attempted it returns with MQRC_ENVIRONMENT_ERROR.

If, at any time during an MQOP_START_WAIT operation there are no registered, non-suspended consumers the call fails with a reason code of MQRC_NO_CALLBACKS_ACTIVE.

If, during an MQOP_START_WAIT operation, the connection is suspended, the MQCTL call returns a warning reason code of MQRC_CONNECTION_SUSPENDED; at this point the connection remains 'started'.

The application can choose to issue MQOP_STOP or MQOP_RESUME. In this instance, the MQOP_RESUME operation blocks.

This option is not supported in a single threaded client.

MQOP_STOP

Stop the consuming of messages, and wait for all consumers to complete their operations before this option completes. This operation releases the connection handle.

If issued from within a callback routine, this option does not take effect until the routine exits. No more message consumer routines are called after the consumer routines for messages already read have completed, and after stop calls (if requested) to callback routines have been made.

If issued outside a callback routine, control does not return to the caller until the consumer routines for messages already read have completed, and after stop calls (if requested) to callbacks have been made. The callbacks themselves, however, remain registered.

This function has no effect on read ahead messages. You must ensure that consumers run MQCLOSE(MQCO_QUIESCE), from within the callback function, to determine whether there are any further messages available to be delivered.

MQOP_SUSPEND

Pause the consuming of messages. This operation releases the connection handle.

This does not have any effect on the reading ahead of messages for the application. If you intend to stop consuming messages for a long period of time, consider closing the queue and reopening it when consumption should continue.

If issued from within a callback routine, it does not take effect until the routine exits. No more message consumer routines will be called after the current routine exits.

If issued outside a callback, control does not return to the caller until the current consumer routine has completed and no more are called.

| **MQOP_RESUME**

| Resume the consuming of messages.

| This option is normally issued from the main application thread, but it can
| also be used from within a callback routine to cancel an earlier suspension
| request issued in the same routine.

| If the MQOP_RESUME is used to resume an MQOP_START_WAIT then
| the operation will block.

| **ControlOpts (MQCTLO) – input**

| Control callback function - ControlOpts parameter

| Options that control the action of MQCTL

| See MQCTLO for details of the structure.

| **CompCode (MQLONG) – output**

| Control callback function -CompCode parameter

| The completion code; it is one of the following:

| **MQCC_OK**

| Successful completion.

| **MQCC_WARNING**

| Warning (partial completion).

| **MQCC_FAILED**

| Call failed.

| **Reason (MQLONG) – output**

| Control callback function - Reason parameter

| The reason codes listed below are the ones that the queue manager can return for
| the *Reason* parameter.

| If *CompCode* is MQCC_OK:

| **MQRC_NONE**

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

| If *CompCode* is MQCC_FAILED:

| **MQRC_ADAPTER_CONV_LOAD_ERROR**

| (2133, X'855') Unable to load data conversion services modules.

| **MQRC_ADAPTER_NOT_AVAILABLE**

| (2204, X'89C') Adapter not available.

| **MQRC_ADAPTER_SERV_LOAD_ERROR**

| (2130, X'852') Unable to load adapter service module.

| **MQRC_API_EXIT_ERROR**

| (2374, X'946') API exit failed.

| **MQRC_API_EXIT_LOAD_ERROR**

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

| **MQRC_ASID_MISMATCH**

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

| **MQRC_BUFFER_LENGTH_ERROR**
 | (2005, X'7D5') Buffer length parameter not valid.

| **MQRC_CALLBACK_LINK_ERROR**
 | (2487, X'9B7') Unable to call the callback routine

| **MQRC_CALLBACK_NOT_REGISTERED**
 | (2448, X'990') Unable to Deregister, Suspend, or Resume because there is
 | no registered callback

| **MQRC_CALLBACK_ROUTINE_ERROR**
 | (2486, X'9B6') Either, both CallbackFunction and CallbackName have been
 | specified on an MQOP_REGISTER call.
 |
 | Or either CallbackFunction or CallbackName have been specified but does
 | not match the currently registered callback function.

| **MQRC_CALLBACK_TYPE_ERROR**
 | (2483, X'9B3') Incorrect CallBackType field.

| **MQRC_CALL_IN_PROGRESS**
 | (2219, X'8AB') MQI call entered before previous call complete.

| **MQRC_CBD_ERROR**
 | (2444, X'98C') Option block is incorrect.

| **MQRC_CBD_OPTIONS_ERROR**
 | (2484, X'9B4') Incorrect MQCBD options field.

| **MQRC_CICS_WAIT_FAILED**
 | (2140, X'85C') Wait request rejected by CICS.

| **MQRC_CONNECTION_BROKEN**
 | (2009, X'7D9') Connection to queue manager lost.

| **MQRC_CONNECTION_NOT_AUTHORIZED**
 | (2217, X'8A9') Not authorized for connection.

| **MQRC_CONNECTION QUIESCING**
 | (2202, X'89A') Connection quiescing.

| **MQRC_CONNECTION_STOPPING**
 | (2203, X'89B') Connection shutting down.

| **MQRC_CORREL_ID_ERROR**
 | (2207, X'89F') Correlation-identifier error.

| **MQRC_GET_INHIBITED**
 | (2016, X'7E0') Gets inhibited for the queue.

| **MQRC_GLOBAL_UOW_CONFLICT**
 | (2351, X'92F') Global units of work conflict.

| **MQRC_GMO_ERROR**
 | (2186, X'88A') Get-message options structure not valid.

| **MQRC_HANDLE_IN_USE_FOR_UOW**
 | (2353, X'931') Handle in use for global unit of work.

| **MQRC_HCONN_ERROR**
 | (2018, X'7E2') Connection handle not valid.

| **MQRC_HOBJ_ERROR**
 | (2019, X'7E3') Object handle not valid.

| **MQRC_INCONSISTENT_BROWSE**
| (2259, X'8D3') Inconsistent browse specification.

| **MQRC_INCONSISTENT_UOW**
| (2245, X'8C5') Inconsistent unit-of-work specification.

| **MQRC_INVALID_MSG_UNDER_CURSOR**
| (2246, X'8C6') Message under cursor not valid for retrieval.

| **MQRC_LOCAL_UOW_CONFLICT**
| (2352, X'930') Global unit of work conflicts with local unit of work.

| **MQRC_MATCH_OPTIONS_ERROR**
| (2247, X'8C7') Match options not valid.

| **MQRC_MAX_MSG_LENGTH_ERROR**
| (2485, X'9B5') Incorrect MaxMsgLength field

| **MQRC_MD_ERROR**
| (2026, X'7EA') Message descriptor not valid.

| **MQRC_MODULE_ENTRY_NOT_FOUND**
| (2497, X'9C1')The specified function entry point could not be found in the
| module.

| **MQRC_MODULE_INVALID**
| (2496, X'9C0') Module is found but is of the wrong type (32bit/64bit) or is
| not a valid dll.

| **MQRC_MODULE_NOT_FOUND**
| (2495, X'9BF') Module not found in the search path or not authorised to
| load.

| **MQRC_MSG_ID_ERROR**
| (2206, X'89E') Message-identifier error.

| **MQRC_MSG_SEQ_NUMBER_ERROR**
| (2250, X'8CA') Message sequence number not valid.

| **MQRC_MSG_TOKEN_ERROR**
| (2331, X'91B') Use of message token not valid.

| **MQRC_NOT_OPEN_FOR_BROWSE**
| (2036, X'7F4') Queue not open for browse.

| **MQRC_NOT_OPEN_FOR_INPUT**
| (2037, X'7F5') Queue not open for input.

| **MQRC_OBJECT_CHANGED**
| (2041, X'7F9') Object definition changed since opened.

| **MQRC_OBJECT_DAMAGED**
| (2101, X'835') Object damaged.

| **MQRC_OPERATION_ERROR**
| (2488, X'9B8') Incorrect Operation code on API Call

| **MQRC_OPTIONS_ERROR**
| (2046, X'7FE') Options not valid or not consistent.

| **MQRC_PAGESET_ERROR**
| (2193, X'891') Error accessing page-set data set.

| **MQRC_Q_DELETED**
| (2052, X'804') Queue has been deleted.

MQRC_Q_INDEX_TYPE_ERROR
 (2394, X'95A') Queue has wrong index type.

MQRC_Q_MGR_NAME_ERROR
 (2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
 (2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_QUIESCING
 (2161, X'871') Queue manager quiescing.

MQRC_Q_MGR_STOPPING
 (2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
 (2102, X'836') Insufficient system resources available.

MQRC_SIGNAL_OUTSTANDING
 (2069, X'815') Signal outstanding for this handle.

MQRC_STORAGE_NOT_AVAILABLE
 (2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
 (2109, X'83D') Call suppressed by exit program.

MQRC_SYNCPOINT_NOT_AVAILABLE
 (2072, X'818') Syncpoint support not available.

MQRC_UNEXPECTED_ERROR
 (2195, X'893') Unexpected error occurred.

MQRC_UOW_ENLISTMENT_ERROR
 (2354, X'932') Enlistment in global unit of work failed.

MQRC_UOW_MIX_NOT_SUPPORTED
 (2355, X'933') Mixture of unit-of-work calls not supported.

MQRC_UOW_NOT_AVAILABLE
 (2255, X'8CF') Unit of work not available for the queue manager to use.

MQRC_WAIT_INTERVAL_ERROR
 (2090, X'82A') Wait interval in MQGMO not valid.

MQRC_WRONG_GMO_VERSION
 (2256, X'8D0') Wrong version of MQGMO supplied.

MQRC_WRONG_MD_VERSION
 (2257, X'8D1') Wrong version of MQMD supplied.

Usage notes for MQCTL

Control callback function - Usage notes

1. Callback routines must check the responses from all services they invoke, and if the routine detects a condition that can not be resolved, it must issue an MQCB MQOP_DEREGISTER command to prevent repeated calls to the callback routine.
2. On z/OS, when Operation is MQOP_START:
 - Programs which use asynchronous callback routines must be authorized to use z/OS UNIX System Services (USS).
 - Language Environment (LE) programs which use asynchronous callback routines must use the LE runtime option POSIX(ON).

- Non-LE programs which use asynchronous callback routines must not use the USS pthread_create interface (callable service BPX1PTC).

Language invocations for MQCTL

Control call backs function - Language invocations

The MQCTL call is supported in the programming languages shown below.

C invocation

MQCTL function call - C language invocation

```
MQCTL (Hconn, Operation, ControlOpts, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHCONN Hconn;      /* Connection handle */
MQLONG  Operation;  /* Operation being processed */
MQCTLO  ControlOpts /* Options that control the action of MQCTL */
MQLONG  CompCode;   /* Completion code */
MQLONG  Reason;     /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCTL' USING HCONN, OPERATION, CTLOPTS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN PIC S9(9) BINARY.
** Operation
01 OPERATION PIC S9(9) BINARY.
** Control Options
01 CTLOPTS.
   COPY CMQCTLOW.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON PIC S9(9) BINARY.
```

PL/I invocation

```
call MQCTL(Hconn, Operation, Ct1Opts, CompCode, Reason)
```

Declare the parameters as follows:

```
dc1 Hconn      fixed bin(31); /* Connection handle */
dc1 Operation  fixed bin(31); /* Operation */
dc1 Ct1Opts like MQCTLO;     /* Options that control the action of MQCTL */
dc1 CompCode   fixed bin(31); /* Completion code */
dc1 Reason     fixed bin(31); /* Reason code qualifying CompCode */
```

MQDISC – Disconnect queue manager

The MQDISC call breaks the connection between the queue manager and the application program, and is the inverse of the MQCONN or MQCONNX call.

- On z/OS, CICS applications do not need to issue this call to disconnect from the queue manager, but might need to issue it to end the use of a connection tag.
- On i5/OS, applications running in compatibility mode do not need to issue this call. See “MQCONN – Connect queue manager” on page 435 for more information.

Syntax for MQDISC

MQDISC (*Hconn*, *CompCode*, *Reason*)

Parameters for MQDISC

The MQDISC call has the following parameters.

Hconn (MQHCONN) – input/output

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on i5/OS for applications running in compatibility mode, you can omit the MQCONN call, and specify the following value for *Hconn*:

MQHC_DEF_HCONN

Default connection handle.

On successful completion of the call, the queue manager sets *Hconn* to a value that is not a valid handle for the environment. This value is:

MQHC_UNUSABLE_HCONN

Unusable connection handle.

On z/OS, *Hconn* is set to a value that is undefined.

CompCode (MQLONG) – output

The completion code; it is one of the following:

MQCC_OK

Successful completion.

MQCC_WARNING

Warning (partial completion).

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

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

If *CompCode* is MQCC_WARNING:

MQRC_BACKED_OUT

(2003, X'7D3') Unit of work backed out.

MQRC_CONN_TAG_NOT_RELEASED

(2344, X'928') Connection tag not released.

MQRC_OUTCOME_PENDING

(2124, X'84C') Result of commit operation is pending.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_DISC_LOAD_ERROR

(2138, X'85A') Unable to load adapter disconnection module.

MQRC_ADAPTER_NOT_AVAILABLE

(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR

(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR

(2374, X'946') API exit failed.

MQRC_API_EXIT_INIT_ERROR

(2375, X'947') API exit initialization failed.

MQRC_API_EXIT_TERM_ERROR

(2376, X'948') API exit termination failed.

MQRC_ASID_MISMATCH

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

MQRC_CALL_IN_PROGRESS

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

MQRC_CONNECTION_BROKEN

(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_STOPPING

(2203, X'89B') Connection shutting down.

MQRC_HCONN_ERROR

(2018, X'7E2') Connection handle not valid.

MQRC_OUTCOME_MIXED

(2123, X'84B') Result of commit or back-out operation is mixed.

MQRC_PAGESET_ERROR

(2193, X'891') Error accessing page-set data set.

MQRC_Q_MGR_NAME_ERROR

(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE

(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_STOPPING

(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM

(2102, X'836') Insufficient system resources available.

MQRC_STORAGE_NOT_AVAILABLE

(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR

(2195, X'893') Unexpected error occurred.

For detailed information on these codes, see:

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

Usage notes for MQDISC

Consider these points when using MQDISC.

1. If an MQDISC call is issued when the connection still has objects open under that connection, the queue manager closes those objects, with the close options set to MQCO_NONE.
2. If the application ends with uncommitted changes in a unit of work, the disposition of those changes depends on how the application ends:
 - a. If the application issues the MQDISC call before ending:
 - For a queue-manager-coordinated unit of work, the queue manager issues the MQCMIT call on behalf of the application. The unit of work is committed if possible, and backed out if not.
 - For an externally-coordinated unit of work, there is no change in the status of the unit of work; however, the queue manager typically indicates that the unit of work must be committed when asked by the unit-of-work coordinator.
On z/OS, CICS, IMS (other than batch DL/1 programs), and RRS applications are like this.
 - b. If the application ends normally but without issuing the MQDISC call, the action taken depends on the environment:
 - On z/OS, the actions described under (a) above occur.
 - In all other cases, the actions described under (c) below occur.
Because of the differences between environments, ensure that applications that you want to port commit or back out the unit of work before they end.
 - c. If the application ends *abnormally* without issuing the MQDISC call, the unit of work is backed out.
3. On z/OS, the following points apply:
 - CICS applications do not have to issue the MQDISC call to disconnect from the queue manager, because the CICS system itself connects to the queue manager, and the MQDISC call has no effect on this connection.
 - CICS, IMS (other than batch DL/1 programs), and RRS applications use units of work that are coordinated by an external unit-of-work coordinator. As a result, the MQDISC call does not affect the status of the unit of work (if any) that exists when the call is issued.
However the MQDISC call *does* indicate the end of use of the connection tag *ConnTag* that was associated with the connection by an earlier MQCONN call issued by the application. If there is an active unit of work that references the connection tag when the MQDISC call is issued, the call completes with completion code MQCC_WARNING and reason code MQRC_CONN_TAG_NOT_RELEASED. The connection tag does not become available for reuse until the external unit-of-work coordinator has resolved the unit of work.
4. On i5/OS, applications running in compatibility mode do not have to issue this call; see the MQCONN call for more details.

Language invocations for MQDISC

The MQDISC call is supported in the programming languages shown below.

C invocation

```
MQDISC (&Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;    /* Connection handle */
MQLONG  CompCode; /* Completion code */
MQLONG  Reason;   /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQDISC' USING HCONN, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle  
01 HCONN PIC S9(9) BINARY.  
** Completion code  
01 COMPCODE PIC S9(9) BINARY.  
** Reason code qualifying COMPCODE  
01 REASON PIC S9(9) BINARY.
```

PL/I invocation

```
call MQDISC (Hconn, CompCode, Reason);
```

Declare the parameters as follows:

```
dc1 Hconn fixed bin(31); /* Connection handle */  
dc1 CompCode fixed bin(31); /* Completion code */  
dc1 Reason fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
CALL MQDISC,(HCONN,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN DS F Connection handle  
COMPCODE DS F Completion code  
REASON DS F Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQDISC Hconn, CompCode, Reason
```

Declare the parameters as follows:

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

MQDLTMH – Delete message handle

The MQDLTMH call deletes a message handle and is the inverse of the MQCRTMH call.

Syntax for MQDLTMH

```
MQDLTMH (Hconn, Hmsg, DltMsgHOpts, CompCode, Reason)
```

Parameters for MQDLTMH

The MQDLTMH call has the following parameters:

Hconn (MQHCONN) – input

This handle represents the connection to the queue manager.

The value must match the connection handle that was used to create the message handle specified in the *Hmsg* parameter.

If the message handle was created using MQHC_UNASSOCIATED_HCONN then a valid connection must be established on the thread deleting the message handle, otherwise the call fails with MQRC_CONNECTION_BROKEN.

Hmsg (MQHMSG) – input/output

This is the message handle to be deleted. The value was returned by a previous MQCRTMH call.

On successful completion of the call, the handle is set to an invalid value for the environment. This value is:

MQHM_UNUSABLE_HMSG
Unusable message handle.

The message handle cannot be deleted if another MQ call is in progress that was passed the same message handle.

DltMsgHOpts (MQDMHO) – input

See MQDMHO for details.

CompCode (MQLONG) – output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason (MQLONG) – output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

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

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'089C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

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

MQRC_CALL_IN_PROGRESS
(2219, X'08AB') MQI call entered before previous call completed.

MQRC_CONNECTION_BROKEN
(2009, X'07D9') Connection to queue manager lost.

MQRC_DMHO_ERROR
(2462, X'099E') Delete message handle options structure not valid.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle pointer not valid.

```

| MQRC_MSG_HANDLE_IN_USE
| (2499, X'09C3') Message handle already in use.
|
| MQRC_OPTIONS_ERROR
| (2046, X'07FE') Options not valid or not consistent.
|
| MQRC_STORAGE_NOT_AVAILABLE
| (2071, X'817') Insufficient storage available.
|
| MQRC_UNEXPECTED_ERROR
| (2195, X'893') Unexpected error occurred.

```

See Chapter 4, "Return codes," on page 661 for more details.

Language invocations for MQDLTMH

The call is supported in the programming languages shown below.

C invocation

Parameters used for the C invocation of MQDLTMH.

```
MQDLTMH (Hconn, &Hmsg, &DltMsgHOpts, &CompCode, &Reason);
```

Declare the parameters as follows:

```

| MQHCONN Hconn; /* Connection handle */
| MQHMSG Hmsg; /* Message handle */
| MQDMHO DltMsgHOpts; /* Options that control the action of MQDLTMH */
| MQLONG CompCode; /* Completion code */
| MQLONG Reason; /* Reason code qualifying CompCode */

```

COBOL invocation

Parameters used for the COBOL invocation of MQDLTMH.

```
CALL 'MQDLTMH' USING HCONN, HMSG, DLTMGOPTS, COMPCODE, REASON.
```

Declare the parameters as follows:

```

| ** Connection handle
| 01 HCONN PIC S9(9) BINARY.
| ** Message handle
| 01 HMSG PIC S9(18) BINARY.
| ** Options that control the action of MQDLTMH
| 01 DLTMGOPTS.
| COPY CMQDMHOV.
| ** Completion code
| 01 COMPCODE PIC S9(9) BINARY.
| ** Reason code qualifying COMPCODE
| 01 REASON PIC S9(9) BINARY.

```

PL/I invocation

Parameters used for the PL/I invocation of MQDLTMH.

```
call MQDLTMH (Hconn, Hmsg, DltMsgHOpts, CompCode, Reason);
```

Declare the parameters as follows:

```

| dcl Hconn fixed bin(31); /* Connection handle */
| dcl Hmsg fixed bin(63); /* Message handle */
| dcl DltMsgHOpts like MQDMHO; /* Options that control the action of MQDLTMH */
| dcl CompCode fixed bin(31); /* Completion code */
| dcl Reason fixed bin(31); /* Reason code qualifying CompCode */

```

System/390 assembler invocation

Parameters used for the System/390 assembler invocation of MQDLTMH.

```
CALL MQDLTMH, (HCONN, HMSG, DLTMGOPTS, COMPCODE, REASON)
```

Declare the parameters as follows:

HCONN	DS	F	Connection handle
HMSG	DS	D	Message handle
DLTMSGHOPTS	CMQDMHOA	,	Options that control the action of MQDLTMH
COMPCODE	DS	F	Completion code
REASON	DS	F	Reason code qualifying COMPCODE

MQDLTMP - Delete message property

The MQDLTMP call deletes a property from a message handle and is the inverse of the MQSETMP call.

Syntax for MQDLTMP

MQDLTMP (*Hconn, Hmsg, DltPropOpts, Name, CompCode, Reason*)

Parameters for MQDLTMP

The MQDLTMP call has the following parameters.

Hconn (MQHCONN) - Input

This handle represents the connection to the queue manager. The value must match the connection handle that was used to create the message handle specified in the *Hmsg* parameter.

If the message handle was created using MQHC_UNASSOCIATED_HCONN then a valid connection must be established on the thread deleting the message handle otherwise the call fails with MQRC_CONNECTION_BROKEN.

Hmsg (MQHMSG) - input

This is the message handle containing the property to be deleted. The value was returned by a previous MQCRTMH call.

DltPropOps (MQDMPO) - Input

See the MQDMPO data type for details.

Name (MQCHARV) - input

The name of the property to delete. See the *Application Programming Guide* for further information on property names.

Wildcards are not allowed in the property name.

CompCode (MQLONG) – output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

| MQCC_FAILED
| Call failed.

| Reason (MQLONG) – output

| The reason code qualifying *CompCode*.

| If *CompCode* is MQCC_OK:

| MQRC_NONE
| (0, X'000') No reason to report.

| If *CompCode* is MQCC_WARNING:

| MQRC_PROPERTY_NOT_AVAILABLE
| (2471, X'09A7') Property not available.

| MQRC_RFH_FORMAT_ERROR
| (2421, X'0975') An MQRFH2 folder containing properties could not be
| parsed.

| If *CompCode* is MQCC_FAILED:

| MQRC_ADAPTER_NOT_AVAILABLE
| (2204, X'089C') Adapter not available.

| MQRC_ADAPTER_SERV_LOAD_ERROR
| (2130, X'0852') Unable to load adapter service module.

| MQRC_ASID_MISMATCH
| (2157, X'086D') Primary and home ASIDs differ.

| MQRC_CALL_IN_PROGRESS
| (2219, X'08AB') MQI call entered before previous call completed.

| MQRC_CONNECTION_BROKEN
| (2009, X'07D9') Connection to queue manager lost.

| MQRC_DMPO_ERROR
| (2481, X'09B1') Delete message property options structure not valid.

| MQRC_HMSG_ERROR
| (2460, X'099C') Message handle not valid.

| MQRC_MSG_HANDLE_IN_USE
| (2499, X'09C3') Message handle already in use.

| MQRC_OPTIONS_ERROR
| (2046, X'07FE') Options not valid or not consistent.

| MQRC_PROPERTY_NAME_ERROR
| (2442, X'098A') Invalid property name.

| MQRC_SOURCE_CCSID_ERROR
| (2111, X'083F') Property name coded character set identifier not valid.

| MQRC_UNEXPECTED_ERROR
| (2195, X'0893') Unexpected error occurred.

| For detailed information on these codes, see:

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

Language invocations for MQDLTMP

C invocation

```
MQDLTMP (Hconn, Hmsg, &DltPropOpts, &Name, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHCONN Hconn;      /* Connection handle */
MQHMSG  Hmsg;       /* Message handle */
MQDMPO  DltPropOpts; /* Options that control the action of MQDLTMP */
MQCHARV Name;      /* Property name */
MQLONG  CompCode;   /* Completion code */
MQLONG  Reason;     /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQDLTMP' USING HCONN, HMSG, DLTPROPOPTS, NAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN  PIC S9(9) BINARY.
** Message handle
01 HMSG  PIC S9(18) BINARY.
** Options that control the action of MQDLTMP
01 DLTPROPOPTS.
   COPY CMQDMPOV.
** Property name
01 NAME
   COPY CMQCHRVA.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON  PIC S9(9) BINARY.
```

PL/I invocation

```
call MQDLTMP (Hconn, Hmsg, DltPropOpts, Name, CompCode, Reason);
```

Declare the parameters as follows:

```
dc1 Hconn      fixed bin(31); /* Connection handle */
dc1 Hmsg       fixed bin(63); /* Message handle */
dc1 DltPropOpts like MQDMPO;  /* Options that control the action of MQDLTMP */
dc1 Name       like MQCHARV;  /* Property name */
dc1 CompCode   fixed bin(31); /* Completion code */
dc1 Reason     fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 assembler invocation

Parameters used for the System/390 assembler invocation of MQDLTMP.

```
CALL MQDLTMP,(HCONN,HMSG,DLTPROPOPTS,NAME,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN      DS      F      Connection handle
HMSG       DS      D      Message handle
DLTPROPOPTS CMQDMPOA ,    Options that control the action of MQDLTMP
NAME       CMQCHRVA ,    Property name
COMPCODE   DS      F      Completion code
REASON     DS      F      Reason code qualifying COMPCODE
```

MQGET – Get message

The MQGET call retrieves a message from a local queue that has been opened using the MQOPEN call.

Syntax for MQGET

MQGET (*Hconn*, *Hobj*, *MsgDesc*, *GetMsgOpts*, *BufferLength*,
Buffer, *DataLength*, *CompCode*, *Reason*)

Parameters for MQGET

The MQGET call has the following parameters.

Hconn (MQHCONN) – input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on i5/OS for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN

Default connection handle.

Hobj (MQHOBJ) – input

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 505 for details):

- MQOO_INPUT_SHARED
- MQOO_INPUT_EXCLUSIVE
- MQOO_INPUT_AS_Q_DEF
- MQOO_BROWSE

MsgDesc (MQMD) – input/output

This structure describes the attributes of the message required, and the attributes of the message retrieved. See “MQMD – Message descriptor” on page 181 for details.

If *BufferLength* is less than the message length, *MsgDesc* is filled by the queue manager, whether or not MQGMO_ACCEPT_TRUNCATED_MSG is specified on the *GetMsgOpts* parameter (see MQGMO - Options field).

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 the fields in the MQMDE have default values, the MQMDE is omitted. A format name of MQFMT_MD_EXTENSION in the *Format* field in MQMD indicates that an MQMDE is present.

The application does not need to provide an MQMD structure provided that a valid message handle is supplied in the *MsgHandle* field. If nothing is provided in this field, the descriptor of the message is taken from the descriptor associated with the message handles.

If MQGMO_PROPERTIES_FORCE_MQRFH2 is specified, or MQGMO_PROPERTIES_AS_Q_DEF is specified and the *PropertyControl* queue attribute is MQPROP_FORCE_MQRFH2, the call fails with reason code MQRC_MD_ERROR.

If match options are specified and the message descriptor associated with the message handle is being used, the input fields used for matching come from the message handle.

GetMsgOpts (MQGMO) – input/output

See “MQGMO – Get-message options” on page 126 for details.

BufferLength (MQLONG) – input

This is the length in bytes of the *Buffer* area. Specify zero for messages that have no data, or if the message is to be removed from the queue and the data discarded (you must specify MQGMO_ACCEPT_TRUNCATED_MSG in this case).

Note: The length of the longest message that it is possible to read from the queue is given by the *MaxMsgLength* queue attribute; see “Attributes for queues” on page 615.

Buffer (MQBYTEExBufferLength) – output

This is the area to contain the message data. Align the buffer on a boundary appropriate to the nature of the data in the message. 4-byte alignment should be suitable for most messages (including messages containing MQ header structures), but some messages might require more stringent alignment. For example, a message containing a 64-bit binary integer might require 8-byte alignment.

If *BufferLength* is less than the message length, as much of the message as possible is moved into *Buffer*; this happens whether or not MQGMO_ACCEPT_TRUNCATED_MSG is specified on the *GetMsgOpts* parameter (see MQGMO - Options field for more information).

The character set and encoding of the data in *Buffer* are given by the *CodedCharSetId* and *Encoding* fields returned in the *MsgDesc* parameter. If these are different from the values required by the receiver, the receiver must convert the application message data to the character set and encoding required. The MQGMO_CONVERT option can be used (with a user-written exit if necessary) to convert the message data; see “MQGMO – Get-message options” on page 126 for details of this option.

Note: All the other parameters on the MQGET call are in the character set and encoding of the local queue manager (given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE, respectively).

If the call fails, the contents of the buffer might still have changed.

In the C programming language, the parameter is declared as a pointer-to-void: the address of any type of data can be specified as the parameter.

If the *BufferLength* parameter is zero, *Buffer* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler can be null.

DataLength (MQLONG) – output

This is the length in bytes of the application data *in the message*. If this is greater than *BufferLength*, only *BufferLength* bytes are returned in the *Buffer* parameter (that is, the message is truncated). If the value is zero, the message contains no application data.

If *BufferLength* is less than the message length, *DataLength* is still filled in by the queue manager, whether or not MQGMO_ACCEPT_TRUNCATED_MSG is specified on the *GetMsgOpts* parameter (see MQGMO - Options field for more information). This allows the application to determine the size of the buffer required to accommodate the message data, and then reissue the call with a buffer of the appropriate size.

However, if the MQGMO_CONVERT option is specified, and the converted message data is too long to fit in *Buffer*, the value returned for *DataLength* is:

- The length of the *unconverted* data, for queue-manager defined formats.
In this case, if the nature of the data causes it to expand during conversion, the application must allocate a buffer somewhat bigger than the value returned by the queue manager for *DataLength*.
- The value returned by the data-conversion exit, for application-defined formats.

CompCode (MQLONG) – output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason (MQLONG) – output

Reason codes that can be returned from an MQGET call.

The reason codes listed below are the ones that the queue manager can return for the *Reason* parameter. If the application specifies the MQGMO_CONVERT option, and a user-written exit is invoked to convert some or all of the message data, the exit decides what value is returned for the *Reason* parameter. As a result, values other than those documented below are possible.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_CONVERTED_MSG_TOO_BIG
(2120, X'848') Converted data too big for buffer.

MQRC_CONVERTED_STRING_TOO_BIG
(2190, X'88E') Converted string too big for field.

MQRC_DBCS_ERROR
(2150, X'866') DBCS string not valid.

MQRC_FORMAT_ERROR
(2110, X'83E') Message format not valid.

MQRC_INCOMPLETE_GROUP
(2241, X'8C1') Message group not complete.

MQRC_INCOMPLETE_MSG
(2242, X'8C2') Logical message not complete.

MQRC_INCONSISTENT_CCSDS
(2243, X'8C3') Message segments have differing CCSIDs.

MQRC_INCONSISTENT_ENCODINGS
(2244, X'8C4') Message segments have differing encodings.

MQRC_INCONSISTENT_UOW
(2245, X'8C5') Inconsistent unit-of-work specification.

MQRC_MSG_TOKEN_ERROR
(2331, X'91B') Invalid use of message token.

MQRC_NO_MSG_LOCKED
(2209, X'8A1') No message locked.

MQRC_NOT_CONVERTED
(2119, X'847') Message data not converted.

MQRC_OPTIONS_CHANGED
(nnnn, X'xxx') Options that were required to be consistent have been changed.

MQRC_PARTIALLY_CONVERTED
(2272, X'8E0') Message data partially converted.

MQRC_SIGNAL_REQUEST_ACCEPTED
(2070, X'816') No message returned (but signal request accepted).

MQRC_SOURCE_BUFFER_ERROR
(2145, X'861') Source buffer parameter not valid.

MQRC_SOURCE_CCSDS_ERROR
(2111, X'83F') Source coded character set identifier not valid.

MQRC_SOURCE_DECIMAL_ENC_ERROR
(2113, X'841') Packed-decimal encoding in message not recognized.

MQRC_SOURCE_FLOAT_ENC_ERROR
(2114, X'842') Floating-point encoding in message not recognized.

MQRC_SOURCE_INTEGER_ENC_ERROR
(2112, X'840') Source integer encoding not recognized.

MQRC_SOURCE_LENGTH_ERROR
(2143, X'85F') Source length parameter not valid.

MQRC_TARGET_BUFFER_ERROR
(2146, X'862') Target buffer parameter not valid.

MQRC_TARGET_CCSDS_ERROR
(2115, X'843') Target coded character set identifier not valid.

MQRC_TARGET_DECIMAL_ENC_ERROR
(2117, X'845') Packed-decimal encoding specified by receiver not recognized.

MQRC_TARGET_FLOAT_ENC_ERROR
(2118, X'846') Floating-point encoding specified by receiver not recognized.

MQRC_TARGET_INTEGER_ENC_ERROR
(2116, X'844') Target integer encoding not recognized.

MQRC_TRUNCATED_MSG_ACCEPTED
(2079, X'81F') Truncated message returned (processing completed).

MQRC_TRUNCATED_MSG_FAILED
(2080, X'820') Truncated message returned (processing not completed).

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_CONV_LOAD_ERROR
(2133, X'855') Unable to load data conversion services modules.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BACKED_OUT
(2003, X'7D3') Unit of work backed out.

MQRC_BUFFER_ERROR
(2004, X'7D4') Buffer parameter not valid.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_STRUC_FAILED
(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CF_STRUC_LIST_HDR_IN_USE
(2347, X'92B') Coupling-facility structure list-header in use.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_CORREL_ID_ERROR
(2207, X'89F') Correlation-identifier error.

MQRC_DATA_LENGTH_ERROR
(2010, X'7DA') Data length parameter not valid.

MQRC_DB2_NOT_AVAILABLE
(2342, X'926') DB2 subsystem not available.

MQRC_GET_INHIBITED
(2016, X'7E0') Gets inhibited for the queue.

MQRC_GLOBAL_UOW_CONFLICT
(2351, X'92F') Global units of work conflict.

MQRC_GMO_ERROR
(2186, X'88A') Get-message options structure not valid.

MQRC_HANDLE_IN_USE_FOR_UOW
(2353, X'931') Handle in use for global unit of work.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_INCONSISTENT_BROWSE
(2259, X'8D3') Inconsistent browse specification.

MQRC_INCONSISTENT_UOW
(2245, X'8C5') Inconsistent unit-of-work specification.

MQRC_INVALID_MSG_UNDER_CURSOR
(2246, X'8C6') Message under cursor not valid for retrieval.

MQRC_LOCAL_UOW_CONFLICT
(2352, X'930') Global unit of work conflicts with local unit of work.

MQRC_MATCH_OPTIONS_ERROR
(2247, X'8C7') Match options not valid.

MQRC_MD_ERROR
(2026, X'7EA') Message descriptor not valid.

MQRC_MSG_ID_ERROR
(2206, X'89E') Message-identifier error.

MQRC_MSG_SEQ_NUMBER_ERROR
(2250, X'8CA') Message sequence number not valid.

MQRC_MSG_TOKEN_ERROR
(2331, X'91B') Use of message token not valid.

MQRC_NO_MSG_AVAILABLE
(2033, X'7F1') No message available.

MQRC_NO_MSG_UNDER_CURSOR
(2034, X'7F2') Browse cursor not positioned on message.

MQRC_NOT_OPEN_FOR_BROWSE
(2036, X'7F4') Queue not open for browse.

MQRC_NOT_OPEN_FOR_INPUT
(2037, X'7F5') Queue not open for input.

MQRC_OBJECT_CHANGED
(2041, X'7F9') Object definition changed since opened.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_Q_DELETED
(2052, X'804') Queue has been deleted.

MQRC_Q_INDEX_TYPE_ERROR
(2394, X'95A') Queue has wrong index type.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR QUIESCING
(2161, X'871') Queue manager quiescing.

MQRC_Q_MGR STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SECOND_MARK_NOT_ALLOWED
(2062, X'80E') A message is already marked.

MQRC_SIGNAL_OUTSTANDING
(2069, X'815') Signal outstanding for this handle.

MQRC_SIGNAL1_ERROR
(2099, X'833') Signal field not valid.

MQRC_STORAGE_MEDIUM_FULL
(2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_SYNCPOINT_LIMIT_REACHED
(2024, X'7E8') No more messages can be handled within current unit of work.

MQRC_SYNCPOINT_NOT_AVAILABLE
(2072, X'818') Syncpoint support not available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

MQRC_UOW_ENLISTMENT_ERROR
(2354, X'932') Enlistment in global unit of work failed.

MQRC_UOW_MIX_NOT_SUPPORTED

(2355, X'933') Mixture of unit-of-work calls not supported.

MQRC_UOW_NOT_AVAILABLE

(2255, X'8CF') Unit of work not available for the queue manager to use.

MQRC_WAIT_INTERVAL_ERROR

(2090, X'82A') Wait interval in MQGMO not valid.

MQRC_WRONG_GMO_VERSION

(2256, X'8D0') Wrong version of MQGMO supplied.

MQRC_WRONG_MD_VERSION

(2257, X'8D1') Wrong version of MQMD supplied.

For detailed information on these codes, see:

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

Usage notes for MQGET

Guidance information for the MQGET call.

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.

The browse options are: MQGMO_BROWSE_FIRST, MQGMO_BROWSE_NEXT, and MQGMO_BROWSE_MSG_UNDER_CURSOR.

2. If the MQGMO_LOCK option is specified with one of the browse options, the browsed message is locked so that it is visible only to this handle.

If the MQGMO_UNLOCK option is specified, a previously-locked message is unlocked. No message is retrieved in this case, and the *MsgDesc*, *BufferLength*, *Buffer*, and *DataLength* parameters are not checked or altered.

3. If the application issuing the MQGET call is running as an MQ client, the message retrieved can be lost if, while processing the MQGET call, the MQ client terminates abnormally or the client connection is severed. This arises because the surrogate that is running on the queue-manager's platform and that issues the MQGET call on the client's behalf cannot detect the loss of the client until the surrogate is about to return the message to the client; this is *after* the message has been removed from the queue. This can occur for both persistent messages and nonpersistent messages.

To eliminate the risk of losing messages in this way, always retrieve messages within units of work (that is, by specifying the MQGMO_SYNCPOINT option on the MQGET call, and using the MQCMIT or MQBACK calls to commit or back out the unit of work when message processing is complete). If MQGMO_SYNCPOINT is specified, and the client terminates abnormally or the connection is severed, the surrogate backs out the unit of work on the queue manager and the message is reinstated on the queue.

In principle, the same situation can arise with applications that are running on the queue-manager's platform, but in this case the window during which a message can be lost is very small. However, as with MQ clients, you can eliminate the risk by retrieving the message within a unit of work.

4. If an application puts a sequence of messages on a particular queue within a single unit of work, and then commits that unit of work successfully, the messages become available for retrieval as follows:

- If the queue is a *nonshared* queue (that is, a local queue), all messages within the unit of work become available at the same time.

- If the queue is a *shared* queue, messages within the unit of work become available in the order in which they were put, but not all at the same time. When the system is heavily laden, it is possible for the first message in the unit of work to be retrieved successfully, but for the MQGET call for the second or subsequent message in the unit of work to fail with MQRC_NO_MSG_AVAILABLE. If this occurs, the application must wait a short while and then try the operation again.
5. If an application puts a sequence of messages on the same queue without using message groups, the order of those messages is preserved provided that certain conditions are satisfied. See MQPUT usage notes for details. If the conditions are satisfied, the messages are presented to the receiving application in the order in which they were sent, provided that:
 - Only one receiver is getting messages from the queue.

If there are two or more applications getting messages from the queue, they must agree with the sender the mechanism to be used to identify messages that belong to a sequence. For example, the sender might set all the *CorrelId* fields in the messages in a sequence to a value that was unique to that sequence of messages.
 - The receiver does not deliberately change the order of retrieval, for example by specifying a particular *MsgId* or *CorrelId*.

If the sending application puts the messages as a message group, the messages are presented to the receiving application in the correct order provided that the receiving application specifies the MQGMO_LOGICAL_ORDER option on the MQGET call. For more information about message groups, see:

 - MQMD - MsgFlags field
 - MQPMO_LOGICAL_ORDER
 - MQGMO_LOGICAL_ORDER

If the user is getting messages in a group under syncpoint, they should ensure the complete group is processed before attempting to finish the transaction.
 6. Applications must test for the feedback code MQFB_QUIT in the *Feedback* field of the *MsgDesc* parameter, and end if they find this value. See MQMD - Feedback field for more information.
 7. If the queue identified by *Hobj* was opened with the MQOO_SAVE_ALL_CONTEXT option, and the completion code from the MQGET call is MQCC_OK or MQCC_WARNING, the context associated with the queue handle *Hobj* is set to the context of the message that has been retrieved (unless the MQGMO_BROWSE_FIRST, MQGMO_BROWSE_NEXT, or MQGMO_BROWSE_MSG_UNDER_CURSOR option is set, in which case the context is marked as not available).

You can use the saved context on a subsequent MQPUT or MQPUT1 call by specifying the MQPMO_PASS_IDENTITY_CONTEXT or MQPMO_PASS_ALL_CONTEXT options. This enables the context of the message received to be transferred in whole or in part to another message (for example, when the message is forwarded to another queue). For more information about message context, see the *WebSphere MQ Application Programming Guide*.
 8. If you include the MQGMO_CONVERT option in the *GetMsgOpts* parameter, the application message data is converted to the representation requested by the receiving application, before the data is placed in the *Buffer* parameter:

- The *Format* field in the control information in the message identifies the structure of the application data, and the *CodedCharSetId* and *Encoding* fields in the control information in the message specify its character-set identifier and encoding.
- The application issuing the MQGET call specifies in the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter the character-set identifier and encoding to which to convert the application message data.

When conversion of the message data is necessary, the conversion is performed either by the queue manager itself or by a user-written exit, depending on the value of the *Format* field in the control information in the message:

- The following format names are formats that are converted by the queue manager; these are called “built-in” formats:
 - MQFMT_ADMIN
 - MQFMT_CICS (z/OS only)
 - MQFMT_COMMAND_1
 - MQFMT_COMMAND_2
 - MQFMT_DEAD_LETTER_HEADER
 - MQFMT_DIST_HEADER
 - MQFMT_EVENT version 1
 - MQFMT_EVENT version 2 (z/OS only)
 - MQFMT_IMS
 - MQFMT_IMS_VAR_STRING
 - MQFMT_MD_EXTENSION
 - MQFMT_PCF
 - MQFMT_REF_MSG_HEADER
 - MQFMT_RF_HEADER
 - MQFMT_RF_HEADER_2
 - MQFMT_STRING
 - MQFMT_TRIGGER
 - MQFMT_WORK_INFO_HEADER (z/OS only)
 - MQFMT_XMIT_Q_HEADER
- The format name MQFMT_NONE is a special value that indicates that the nature of the data in the message is undefined. As a consequence, the queue manager does not attempt conversion when the message is retrieved from the queue.

Note: If MQGMO_CONVERT is specified on the MQGET call for a message that has a format name of MQFMT_NONE, and the character set or encoding of the message differs from that specified in the *MsgDesc* parameter, the message is returned in the *Buffer* parameter (assuming no other errors), but the call completes with completion code MQCC_WARNING and reason code MQRC_FORMAT_ERROR.

You can use MQFMT_NONE 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 to send the message data.

- All other format names pass the message 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 must not begin with the letters MQ.

See Chapter 8, “Data conversion,” on page 677 for details of the data-conversion exit.

User data in the message can be converted between any supported character sets and encodings. However, be aware that, if the message contains one or more MQ header structures, the message cannot be converted from or to a character set that has double-byte or multi-byte characters for any of the characters that are valid in queue names. Reason code MQRC_SOURCE_CCSDID_ERROR or MQRC_TARGET_CCSDID_ERROR results if this is attempted, and the message is returned unconverted. Unicode character set UCS-2 is an example of such a character set.

On return from MQGET, the following reason code indicates that the message was converted successfully:

- MQRC_NONE

The following reason code indicates that the message *might* have been converted successfully; the application must check the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter to find out:

- MQRC_TRUNCATED_MSG_ACCEPTED

All other reason codes indicate that the message was not converted.

Note: The interpretation of this reason code is true for conversions performed by a user-written exit *only* if the exit conforms to the processing guidelines described in Chapter 8, “Data conversion,” on page 677.

9. When using the object-oriented interface to get messages, you can choose not to specify a buffer to hold the message data for an MQGET call. In previous versions of WebSphere MQ it was possible for MQGET to fail with reason code MQRC_CONVERTED_MSG_TO_BIG, however, even when a buffer was not specified. In WebSphere MQ Version 7, when you get a message using an object-oriented application without restricting the size of the receive message buffer, the application does not fail with MQRC_CONVERTED_MSG_TOO_BIG, and receives the converted message. This is true of the following environments:

- .NET, including fully managed applications
- C++
- Java (WebSphere MQ classes for Java)

Note: For all clients, if the value of *sharingConversations* is zero, the channel operates as it did before WebSphere MQ Version 7.0, and message handling reverts to Version 6 behavior. In this situation, if the buffer is too small to receive the converted message, the unconverted message is returned, with reason code MQRC_CONVERTED_MSG_TOO_BIG. For more information about *sharingConversations*, see Implications of sharing conversations.

10. For the built-in formats, the queue manager can perform *default conversion* of character strings in the message when the MQGMO_CONVERT option is specified. Default conversion allows the queue manager to use an installation-specified default character set that approximates the actual character set, when converting string data. As a result, the MQGET call can

succeed with completion code MQCC_OK, instead of completing with MQCC_WARNING and reason code MQRC_SOURCE_CCSID_ERROR or MQRC_TARGET_CCSID_ERROR.

Note: The result of using an approximate character set to convert string data is that some characters might be converted incorrectly. To avoid this, use characters in the string that 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* the following are true:
 - The application specifies MQGMO_CONVERT.
 - The message contains data that must be converted either from or to a character set that is not supported.
 - Default conversion was enabled when the queue manager was installed or restarted.
- Default conversion of the character fields in the MQMD and MQMDE structures occurs as necessary, provided that default conversion is enabled for the queue manager. The conversion is performed even if the MQGMO_CONVERT option is not specified by the application on the MQGET call.

11. For the Visual Basic programming language, the following points apply:

- If the size of the *Buffer* parameter is less than the length specified by the *BufferLength* parameter, the call fails with reason code MQRC_STORAGE_NOT_AVAILABLE.
- The *Buffer* parameter is declared as being of type String. If the data to be retrieved from the queue is not of type String, use the MQGETAny call in place of MQGET.

The MQGETAny call has the same parameters as the MQGET call, except that the *Buffer* parameter is declared as being of type Any, allowing any type of data to be retrieved. However, this means that *Buffer* cannot be checked to ensure that it is at least *BufferLength* bytes in size.

12. Not all MQGET options are supported when read ahead is enabled. The following table indicated which options are allowed and whether they can be altered between MQGET calls.

Table 89. MQGET options permitted when read ahead is enabled

	Permitted when read ahead is enabled and can be altered between MQGET calls	Permitted when read ahead is enabled but cannot be altered between MQGET calls ^a	MQGET options that are not permitted when read ahead is enabled ^b
MQGET MD values	MsgId ^c CorrelId ^c	Encoding CodedCharSetId	
MQGET MQGMO options	MQGMO_WAIT MQGMO_NO_WAIT MQGMO_FAIL_IF QUIESCING MQGMO_BROWSE_FIRST ^d MQGMO_BROWSE_NEXT ^d MQGMO_BROWSE_MESSAGE_UNDER_CURSOR ^d	MQGMO_SYNCPOINT_IF_PERSISTENT MQGMO_NO_SYNCPOINT MQGMO_ACCEPT_TRUNCATED_MSG MQGMO_CONVERT MQGMO_LOGICAL_ORDER MQGMO_COMPLETE_MSG MQGMO_ALL_MSGS_AVAILABLE MQGMO_ALL_SEGMENTS_AVAILABLE MQGMO_MARK_BROWSE_HANDLE MQGMO_MARK_BROWSE_CO_OP MQGMO_UNMARK_BROWSE_CO_OP MQGMO_UNMARK_BROWSE_HANDLE MQGMO_UNMARKED_BROWSE_MSG MQGMO_PROPERTIES_FORCE_MQRFH2 MQGMO_NO_PROPERTIES MQGMO_PROPERTIES_IN_HANDLE MQGMO_PROPERTIES_COMPATIBILITY	MQGMO_SET_SIGNAL MQGMO_SYNCPOINT MQGMO_MARK_SKIP_BACKOUT MQGMO_MSG_UNDER_CURSOR ^d MQGMO_LOCK MQGMO_UNLOCK
MQGMO values		MsgHandle	

- a. If these options are altered between MQGET calls an MQRC_OPTIONS_CHANGED reason code will be returned.

- b. If these options are specified on the first MQGET call then read ahead will be disabled. If these options are specified on a subsequent MQGET call a reason code MQRC_OPTIONS_ERROR will be returned.
 - c. The client applications needs to be aware that if the MsgId and CorrelId values are altered between MQGET calls messages with the previous values may have already been sent to the client and will remain in the client read ahead buffer until consumed (or automatically purged).
 - d. The first MQGET call determines whether messages are to be browsed or got from a queue when read ahead is enabled. If the application attempts to use a combination of browse and get an MQRC_OPTIONS_CHANGED reason code is returned.
 - e. MQGMO_MSG_UNDER_CURSOR is not possible with read ahead. Messages can be browsed or got when read ahead is enabled but not a combination of both.
13. Applications can destructively get uncommitted messages only if those messages were put in the same local unit of work as the get. Applications cannot get uncommitted messages nondestructively.
14. Messages under a browse cursor can be retrieved in a unit of work. It is not possible to retrieve an uncommitted message in this way.

Language invocations for MQGET

The MQGET call is supported in the programming languages shown below.

C invocation

```
MQGET (Hconn, Hobj, &MsgDesc, &GetMsgOpts, BufferLength, Buffer,
      &DataLength, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;          /* Connection handle */
MQHOBJ  Hobj;           /* Object handle */
MQMD    MsgDesc;       /* Message descriptor */
MQGMO   GetMsgOpts;    /* Options that control the action of MQGET */
MQLONG  BufferLength;   /* Length in bytes of the Buffer area */
MQBYTE  Buffer[n];     /* Area to contain the message data */
MQLONG  DataLength;    /* Length of the message */
MQLONG  CompCode;     /* Completion code */
MQLONG  Reason;       /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQGET' USING HCONN, HOBJ, MSGDESC, GETMSGOPTS, BUFFERLENGTH,
                  BUFFER, DATALENGTH, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN          PIC S9(9) BINARY.
** Object handle
01 HOBJ           PIC S9(9) BINARY.
** Message descriptor
01 MSGDESC.
   COPY CMQMDV.
** Options that control the action of MQGET
01 GETMSGOPTS.
   COPY CMQGMV.
** Length in bytes of the BUFFER area
01 BUFFERLENGTH  PIC S9(9) BINARY.
** Area to contain the message data
01 BUFFER        PIC X(n).
** Length of the message
```

```

01 DATALENGTH PIC S9(9) BINARY.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON PIC S9(9) BINARY.

```

PL/I invocation

```
call MQGET (Hconn, Hobj, MsgDesc, GetMsgOpts, BufferLength, Buffer,
           DataLength, CompCode, Reason);
```

Declare the parameters as follows:

```

dcl Hconn          fixed bin(31); /* Connection handle */
dcl Hobj           fixed bin(31); /* Object handle */
dcl MsgDesc        like MQMD;    /* Message descriptor */
dcl GetMsgOpts     like MQGMO;   /* Options that control the action of
                                MQGET */
dcl BufferLength    fixed bin(31); /* Length in bytes of the Buffer
                                area */
dcl Buffer          char(n);      /* Area to contain the message data */
dcl DataLength     fixed bin(31); /* Length of the message */
dcl CompCode       fixed bin(31); /* Completion code */
dcl Reason         fixed bin(31); /* Reason code qualifying CompCode */

```

System/390 assembler invocation

```
CALL MQGET,(HCONN,HOBJ,MSGDESC,GETMSGOPTS,BUFFERLENGTH,
           BUFFER,DATALENGTH,COMPCODE,REASON)
```

Declare the parameters as follows:

```

HCONN      DS      F      Connection handle
HOBJ       DS      F      Object handle
MSGDESC    CMQMDA  ,      Message descriptor
GETMSGOPTS CMQGMOA ,      Options that control the action of MQGET
BUFFERLENGTH DS    F      Length in bytes of the BUFFER area
BUFFER     DS      CL(n) Area to contain the message data
DATALENGTH DS      F      Length of the message
COMPCODE   DS      F      Completion code
REASON     DS      F      Reason code qualifying COMPCODE

```

Visual Basic invocation

```
MQGET Hconn, Hobj, MsgDesc, GetMsgOpts, BufferLength, Buffer,
      DataLength, CompCode, Reason
```

Declare the parameters as follows:

```

Dim Hconn      As Long 'Connection handle'
Dim Hobj       As Long 'Object handle'
Dim MsgDesc    As MQMD 'Message descriptor'
Dim GetMsgOpts As MQGMO 'Options that control the action of MQGET'
Dim BufferLength As Long 'Length in bytes of the Buffer area'
Dim Buffer      As String 'Area to contain the message data'
Dim DataLength As Long 'Length of the message'
Dim CompCode   As Long 'Completion code'
Dim Reason     As Long 'Reason code qualifying CompCode'

```

MQINQ – Inquire object attributes

The MQINQ call returns an array of integers and a set of character strings containing the attributes of an object.

The following types of object are valid:

- Queue manager
- Queue

- Namelist
- Process definition

Syntax for MQINQ

MQINQ (*Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs, CharAttrLength, CharAttrs, CompCode, Reason*)

Parameters for MQINQ

The MQINQ call has the following parameters.

Hconn (MQHCONN) – input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on i5/OS for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Hobj (MQHOBJ) – input

This handle represents the object (of any type) whose attributes are required. The handle must have been returned by a previous MQOPEN call that specified the MQOO_INQUIRE option.

SelectorCount (MQLONG) – input

This is the count of selectors that are supplied in the *Selectors* array. It is the number of attributes that are to be returned. Zero is a valid value. The maximum number allowed is 256.

Selectors (MQLONGxSelectorCount) – input

This is an array of *SelectorCount* attribute selectors; each selector identifies an attribute (integer or character) whose value is required.

Each selector must be valid for the type of object that *Hobj* represents, otherwise the call fails with completion code MQCC_FAILED and reason code MQRC_SELECTOR_ERROR.

In the special case of queues:

- If the selector is not valid for queues of *any* type, the call fails with completion code MQCC_FAILED and reason code MQRC_SELECTOR_ERROR.
- If the selector applies *only* to queues of type or types other than that of the object, the call succeeds with completion code MQCC_WARNING and reason code MQRC_SELECTOR_NOT_FOR_TYPE.
- If the queue being inquired is a cluster queue, the selectors that are valid depend on how the queue was resolved; see “Usage notes for MQINQ” on page 492 for further details.

You can specify selectors in any order. Attribute values that correspond to integer attribute selectors (MQIA_* selectors) are returned in *IntAttrs* in the same order in

which these selectors occur in *Selectors*. Attribute values that correspond to character attribute selectors (MQCA_* selectors) are returned in *CharAttrs* in the same order in which those selectors occur. MQIA_* selectors can be interleaved with the MQCA_* selectors; only the relative order within each type is important.

Note:

1. The integer and character attribute selectors are allocated within two different ranges; the MQIA_* selectors reside within the range MQIA_FIRST through MQIA_LAST, and the MQCA_* selectors within the range MQCA_FIRST through MQCA_LAST.
For each range, the constants MQIA_LAST_USED and MQCA_LAST_USED define the highest value that the queue manager will accept.
2. If all of the MQIA_* selectors occur first, the same element numbers can be used to address corresponding elements in the *Selectors* and *IntAttrs* arrays.
3. If the *SelectorCount* parameter is zero, *Selectors* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler might be null.

The attributes that can be inquired are listed in the following tables. For the MQCA_* selectors, the constant that defines the length in bytes of the resulting string in *CharAttrs* is given in parentheses.

The tables that follow list the selectors, by object, in alphabetic order, as follows:

- Table 90 MQINQ attribute selectors for queues
- Table 91 on page 484 MQINQ attribute selectors for namelists
- Table 92 on page 484 MQINQ attribute selectors for process definitions
- Table 93 on page 485 MQINQ attribute selectors for the queue manager

All selectors are supported on all WebSphere MQ platforms, except where indicated in the **Note** column as follows:

Not z/OS

Supported on all platforms **except** z/OS

z/OS Supported **only** on z/OS

Table 90. MQINQ attribute selectors for queues

Selector	Description	Note
MQCA_ALTERATION_DATE	Date of most-recent alteration (MQ_DATE_LENGTH)	
MQCA_ALTERATION_TIME	Time of most-recent alteration (MQ_TIME_LENGTH)	
MQCA_BACKOUT_REQ_Q_NAME	Excessive backout requeue name (MQ_Q_NAME_LENGTH)	
MQCA_BASE_Q_NAME	Name of queue that alias resolves to (MQ_Q_NAME_LENGTH)	
MQCA_CF_STRUC_NAME	Coupling-facility structure name (MQ_CF_STRUC_NAME_LENGTH)	z/OS
MQCA_CLUSTER_NAME	Cluster name (MQ_CLUSTER_NAME_LENGTH)	
MQCA_CLUSTER_NAMELIST	Cluster namelist (MQ_NAMELIST_NAME_LENGTH)	
MQCA_CREATION_DATE	Queue creation date (MQ_CREATION_DATE_LENGTH)	
MQCA_CREATION_TIME	Queue creation time (MQ_CREATION_TIME_LENGTH)	

Table 90. MQINQ attribute selectors for queues (continued)

Selector	Description	Note
MQCA_INITIATION_Q_NAME	Initiation queue name (MQ_Q_NAME_LENGTH)	
MQCA_PROCESS_NAME	Name of process definition (MQ_PROCESS_NAME_LENGTH)	
MQCA_Q_DESC	Queue description (MQ_Q_DESC_LENGTH)	
MQCA_Q_NAME	Queue name (MQ_Q_NAME_LENGTH)	
MQCA_REMOTE_Q_MGR_NAME	Name of remote queue manager (MQ_Q_MGR_NAME_LENGTH)	
MQCA_REMOTE_Q_NAME	Name of remote queue as known on remote queue manager (MQ_Q_NAME_LENGTH)	
MQCA_STORAGE_CLASS	Name of storage class (MQ_STORAGE_CLASS_LENGTH)	z/OS
MQCA_TRIGGER_DATA	Trigger data (MQ_TRIGGER_DATA_LENGTH)	
MQCA_XMIT_Q_NAME	Transmission queue name (MQ_Q_NAME_LENGTH)	
MQIA_ACCOUNTING_Q	Controls collection of accounting data for queue	Not z/OS
MQIA_BACKOUT_THRESHOLD	Backout threshold	
MQIA_CLWL_Q_PRIORITY	Priority of queue	
MQIA_CLWL_Q_RANK	Rank of queue	
MQIA_CLWL_USEQ	Use remote queues	
MQIA_CURRENT_Q_DEPTH	Number of messages on queue	
MQIA_DEF_BIND	Default binding	
MQIA_DEF_INPUT_OPEN_OPTION	Default open-for-input option	
MQIA_DEF_PERSISTENCE	Default message persistence	
MQIA_DEF_PRIORITY	Default message priority	
MQIA_DEFINITION_TYPE	Queue definition type	
MQIA_DIST_LISTS	Distribution list support	Not z/OS
MQIA_HARDEN_GET_BACKOUT	Whether to harden backout count	
MQIA_INDEX_TYPE	Type of index maintained for queue	z/OS
MQIA_INHIBIT_GET	Whether get operations are allowed	
MQIA_INHIBIT_PUT	Whether put operations are allowed	
MQIA_MAX_MSG_LENGTH	Maximum message length	
MQIA_MAX_Q_DEPTH	Maximum number of messages allowed on queue	
MQIA_MSG_DELIVERY_SEQUENCE	Whether message priority is relevant	
MQIA_NPM_CLASS	Level of reliability for nonpersistent messages	
MQIA_OPEN_INPUT_COUNT	Number of MQOPEN calls that have the queue open for input	
MQIA_OPEN_OUTPUT_COUNT	Number of MQOPEN calls that have the queue open for output	
MQIA_PROPERTY_CONTROL	Property control attribute	
MQIA_Q_DEPTH_HIGH_EVENT	Control attribute for queue depth high events	Not z/OS
MQIA_Q_DEPTH_HIGH_LIMIT	High limit for queue depth	Not z/OS
MQIA_Q_DEPTH_LOW_EVENT	Control attribute for queue depth low events	Not z/OS
MQIA_Q_DEPTH_LOW_LIMIT	Low limit for queue depth	Not z/OS

Table 90. MQINQ attribute selectors for queues (continued)

Selector	Description	Note
MQIA_Q_DEPTH_MAX_EVENT	Control attribute for queue depth max events	Not z/OS
MQIA_Q_SERVICE_INTERVAL	Limit for queue service interval	Not z/OS
MQIA_Q_SERVICE_INTERVAL_EVENT	Control attribute for queue service interval events	Not z/OS
MQIA_Q_TYPE	Queue type	
MQIA_QSG_DISP	Queue-sharing group disposition	z/OS
MQIA_RETENTION_INTERVAL	Queue retention interval	
MQIA_SCOPE	Queue definition scope	Not z/OS
MQIA_SHAREABILITY	Whether queue can be shared for input	
MQIA_STATISTICS_Q	Controls collection of statistics data for queue	Not z/OS
MQIA_TRIGGER_CONTROL	Trigger control	
MQIA_TRIGGER_DEPTH	Trigger depth	
MQIA_TRIGGER_MSG_PRIORITY	Threshold message priority for triggers	
MQIA_TRIGGER_TYPE	Trigger type	
MQIA_USAGE	Usage	

Table 91. MQINQ attribute selectors for namelists

Selector	Description	Note
MQCA_ALTERATION_DATE	Date of most-recent alteration (MQ_DATE_LENGTH)	
MQCA_ALTERATION_TIME	Time of most-recent alteration (MQ_TIME_LENGTH)	
MQCA_NAMELIST_DESC	Namelist description (MQ_NAMELIST_DESC_LENGTH)	
MQCA_NAMELIST_NAME	Name of namelist object (MQ_NAMELIST_NAME_LENGTH)	
MQIA_NAMELIST_TYPE	Namelist type	z/OS
MQCA_NAMES	Names in the namelist (MQ_Q_NAME_LENGTH × Number of names in the list)	
MQIA_NAME_COUNT	Number of names in the namelist	
MQIA_QSG_DISP	Queue-sharing group disposition	z/OS

Table 92. MQINQ attribute selectors for process definitions

Selector	Description	Note
MQCA_ALTERATION_DATE	Date of most-recent alteration (MQ_DATE_LENGTH)	
MQCA_ALTERATION_TIME	Time of most-recent alteration (MQ_TIME_LENGTH)	
MQCA_APPL_ID	Application identifier (MQ_PROCESS_APPL_ID_LENGTH)	
MQCA_ENV_DATA	Environment data (MQ_PROCESS_ENV_DATA_LENGTH)	
MQCA_PROCESS_DESC	Description of process definition (MQ_PROCESS_DESC_LENGTH)	
MQCA_PROCESS_NAME	Name of process definition (MQ_PROCESS_NAME_LENGTH)	
MQCA_USER_DATA	User data (MQ_PROCESS_USER_DATA_LENGTH)	

Table 92. MQINQ attribute selectors for process definitions (continued)

Selector	Description	Note
MQIA_APPL_TYPE	Application type	
MQIA_QSG_DISP	Queue-sharing group disposition	z/OS

Table 93. MQINQ attribute selectors for the queue manager

Selector	Description	Note
MQCA_ALTERATION_DATE	Date of most-recent alteration (MQ_DATE_LENGTH)	
MQCA_ALTERATION_TIME	Time of most-recent alteration (MQ_TIME_LENGTH)	
MQCA_CHANNEL_AUTO_DEF_EXIT	Automatic channel definition exit name (MQ_EXIT_NAME_LENGTH)	
MQCA_CHINIT_SERVICE_PARM	Reserved for use by IBM	
MQCA_CLUSTER_WORKLOAD_DATA	Data passed to cluster workload exit (MQ_EXIT_DATA_LENGTH)	
MQCA_CLUSTER_WORKLOAD_EXIT	Name of cluster workload exit (MQ_EXIT_NAME_LENGTH)	
MQCA_COMMAND_INPUT_Q_NAME	System command input queue name (MQ_Q_NAME_LENGTH)	
MQCA_DEAD_LETTER_Q_NAME	Name of dead-letter queue (MQ_Q_NAME_LENGTH)	
MQCA_DEF_XMIT_Q_NAME	Default transmission queue name (MQ_Q_NAME_LENGTH)	
MQCA_DNS_GROUP	Name of the group for the TCP listener that handles inbound transmissions for the queue-sharing group to join when using Workload Manager Dynamic Domain Name Services support	z/OS
MQCA_IGQ_USER_ID	Intra-group queuing user identifier (MQ_USER_ID_LENGTH)	z/OS
MQCA_LU_GROUP_NAME	Generic LU name for the LU 6.2 listener that handles inbound transmissions for the queue-sharing group to use	z/OS
MQCA_LU_NAME	Name of the LU to use for outbound LU 6.2 transmissions. Set this to the same LU that the listener uses for inbound transmissions	z/OS
MQCA_LU62_ARM_SUFFIX	Suffix of the SYS1.PARMLIB member APPCPMxx, that nominates the LUADD for this channel initiator	z/OS
MQCA_PARENT	Name of a hierarchically connected queue manager that is nominated as the parent of this queue manager	
MQCA_Q_MGR_DESC	Queue manager description (MQ_Q_MGR_DESC_LENGTH)	
MQCA_Q_MGR_IDENTIFIER	Queue-manager identifier (MQ_Q_MGR_IDENTIFIER_LENGTH)	
MQCA_Q_MGR_NAME	Name of local queue manager (MQ_Q_MGR_NAME_LENGTH)	

|
|
|

Table 93. MQINQ attribute selectors for the queue manager (continued)

Selector	Description	Note
MQCA_QSG_NAME	Queue-sharing group name (MQ_QSG_NAME_LENGTH)	z/OS
MQCA_REPOSITORY_NAME	Name of cluster for which queue manager provides repository services (MQ_Q_MGR_NAME_LENGTH)	
MQCA_REPOSITORY_NAMELIST	Name of namelist object containing names of clusters for which queue manager provides repository services (MQ_NAMELIST_NAME_LENGTH)	
MQIA_ACCOUNTING_CONN_OVERRIDE	Override accounting settings	Not z/OS
MQIA_ACCOUNTING_INTERVAL	How often to write intermediate accounting records	Not z/OS
MQIA_ACCOUNTING_MQI	Controls collection of accounting information for MQI data	Not z/OS
MQIA_ACCOUNTING_Q	Controls collection of accounting information for queues	Not z/OS
MQIA_ACTIVE_CHANNELS	Maximum number of channels that can be active at any one time	z/OS
MQIA_ADOPTNEWMCA_CHECK	Elements checked to determine whether to adopt an MCA when a new inbound channel is detected that has the same name as an MCA that is already active	z/OS
MQIA_ADOPTNEWMCA_INTERVAL	Amount of time, in seconds, that the new channel waits for the orphaned channel to end	Not z/OS
MQIA_ADOPTNEWMCA_TYPE	Whether to restart an orphaned instance of an MCA of a given channel type automatically when a new inbound channel request matching the AdoptNewMCACheck parameters is detected	z/OS
MQIA_AUTHORITY_EVENT	Control attribute for authority events	Not z/OS
MQIA_BRIDGE_EVENT	Control attribute for IMS bridge events	z/OS
MQIA_CHANNEL_AUTO_DEF	Control attribute for automatic channel definition	Not z/OS
MQIA_CHANNEL_AUTO_DEF_EVENT	Control attribute for automatic channel definition events	Not z/OS
MQIA_CHANNEL_EVENT	Control attribute for channel events	
MQIA_CHINIT_ADAPTERS	Number of adapter subtasks to use for processing WebSphere MQ calls	z/OS
MQIA_CHINIT_DISPATCHERS	Number of dispatchers to use for the channel initiator	z/OS
MQIA_CHINIT_TRACE_AUTO_START	Whether to start channel initiator trace automatically	z/OS
MQIA_CHINIT_TRACE_TABLE_SIZE	Size of the channel initiator's trace data space (in MB)	z/OS
MQIA_CLUSTER_WORKLOAD_LENGTH	Cluster workload length.	
MQIA_CLWL_MRU_CHANNELS	Number of most recently used channels for cluster workload balancing	
MQIA_CLWL_USEQ	Use remote queues	

Table 93. MQINQ attribute selectors for the queue manager (continued)

Selector	Description	Note
MQIA_CODED_CHAR_SET_ID	Coded character set identifier	
MQIA_COMMAND_EVENT	Control attribute for command events	
MQIA_COMMAND_LEVEL	Command level supported by queue manager	
MQIA_CONFIGURATION_EVENT	Control attribute for configuration events	Not z/OS
MQIA_DIST_LISTS	Distribution list support	Not z/OS
MQIA_DNS_WLM	Whether the TCP listener that handles inbound transmissions for the queue-sharing group registers with Workload Manager for Dynamic Domain Name Services	z/OS
MQIA_EXPIRY_INTERVAL	Interval between scans for expired messages	z/OS
MQIA_IGQ_PUT_AUTHORITY	Intra-group queuing put authority	z/OS
MQIA_INHIBIT_EVENT	Control attribute for inhibit events	Not z/OS
MQIA_INTRA_GROUP_QUEUING	Intra-group queuing support	z/OS
MQIA_LISTENER_TIMER	Time interval (in seconds) between WebSphere MQ attempts to restart the listener if there has been an APPC or TCP/IP failure	z/OS
MQIA_LOCAL_EVENT	Control attribute for local events	Not z/OS
MQIA_LOGGER_EVENT	Control attribute for inhibit events	Not z/OS
MQIA_LU62_CHANNELS	Maximum number of channels that can be current, or clients that can be connected, using the LU 6.2 transmission protocol	z/OS
MQIA_MSG_MARK_BROWSE_INTERVAL	Time interval (in milliseconds) after which the queue manager can automatically remove a mark from browse messages	
MQIA_MAX_CHANNELS	Maximum number of channels that can be current (including server-connection channels with connected clients)	z/OS
MQIA_MAX_HANDLES	Maximum number of handles	
MQIA_MAX_MSG_LENGTH	Maximum message length	
MQIA_MAX_PRIORITY	Maximum priority	
MQIA_MAX_UNCOMMITTED_MSGS	Maximum number of uncommitted messages within a unit of work	
MQIA_OUTBOUND_PORT_MAX	With MQIA_OUTBOUND_PORT_MIN, defines range of port numbers to use when binding outgoing channels	z/OS
MQIA_OUTBOUND_PORT_MIN	With MQIA_OUTBOUND_PORT_MAX, defines range of port numbers to use when binding outgoing channels	z/OS
MQIA_PERFORMANCE_EVENT	Control attribute for performance events	Not z/OS
MQIA_PLATFORM	Platform on which the queue manager resides	
MQIA_PUBSUB_MAXMSG_RETRY_COUNT	The number of retries when processing (under syncpoint) a failed command message	

Table 93. MQINQ attribute selectors for the queue manager (continued)

Selector	Description	Note
MQIA_PUBSUB_MODE	Whether the publish/subscribe engine and the queued publish/subscribe interface are running, therefore allowing applications to publish or subscribe using the application programming interface and the queues that are monitored by the queued publish/subscribe interface	
MQIA_PUBSUB_NP_MSG	Whether to discard (or keep) an undelivered input message	
MQIA_PUBSUB_NP_RESP	Controls the behavior of undelivered response messages	
MQIA_PUBSUB_SYNC_PT	Whether only persistent (or all) messages should be processed under syncpoint	
MQIA_RECEIVE_TIMEOUT	Approximately how long a TCP/IP channel waits to receive data, including heartbeats, from its partner, before returning to the inactive state. This is the numeric value qualified by MQIA_RECEIVE_TIMEOUT_TYPE.	z/OS
MQIA_RECEIVE_TIMEOUT_MIN	Minimum time that a TCP/IP channel waits to receive data, including heartbeats, from its partner, before returning to the inactive state	z/OS
MQIA_RECEIVE_TIMEOUT_TYPE	Approximately how long a TCP/IP channel waits to receive data, including heartbeats, from its partner, before returning to the inactive state. This is the qualifier applied to MQIA_RECEIVE_TIMEOUT.	z/OS
MQIA_REMOTE_EVENT	Control attribute for remote events	Not z/OS
MQIA_SECURITY_CASE	Case of security profiles	z/OS
MQIA_SSL_EVENT	Control attribute for channel events	
MQIA_SSL_FIPS_REQUIRED	Use only FIPS-certified algorithms for cryptography	
MQIA_SSL_RESET_COUNT	SSL key reset count	
MQIA_START_STOP_EVENT	Control attribute for start stop events	Not z/OS
MQIA_STATISTICS_AUTO_CLUSSDR	Controls collection of statistics monitoring information for cluster sender channels	Not z/OS
MQIA_STATISTICS_CHANNEL	Controls collection of statistics data for channels	Not z/OS
MQIA_STATISTICS_INTERVAL	How often to write statistics monitoring data	Not z/OS
MQIA_STATISTICS_MQI	Controls collection of statistics monitoring information for queue manager	Not z/OS
MQIA_STATISTICS_Q	Controls collection of statistics data for queues	Not z/OS
MQIA_SYNCPOINT	Syncpoint availability	
MQIA_TCP_CHANNELS	Maximum number of channels that can be current, or clients that can be connected, using the TCP/IP transmission protocol	z/OS
MQIA_TCP_KEEP_ALIVE	Whether to use the TCP KEEPALIVE facility to check that the other end of the connection is still available	z/OS
MQIA_TCP_NAME	Name of the TCP/IP system that you are using	z/OS

Table 93. MQINQ attribute selectors for the queue manager (continued)

Selector	Description	Note
MQIA_TCP_STACK_TYPE	Whether the channel initiator can use only the TCP/IP address space specified in TCPNAME, or can optionally bind to any selected TCP/IP address	z/OS
MQIA_TRACE_ROUTE_RECORDING	Controls recording of trace-route information	z/OS
MQIA_TREE_LIFE_TIME	Lifetime of unused non-administrative topics	
MQIA_TRIGGER_INTERVAL	Trigger interval	

IntAttrCount (MQLONG) – input

This is the number of elements in the *IntAttrs* array. Zero is a valid value.

If this is at least the number of MQIA_* selectors in the *Selectors* parameter, all integer attributes requested are returned.

IntAttrs (MQLONGxIntAttrCount) – output

This is an array of *IntAttrCount* integer attribute values.

Integer attribute values are returned in the same order as the MQIA_* selectors in the *Selectors* parameter. If the array contains more elements than the number of MQIA_* selectors, the excess elements are unchanged.

If *Hobj* represents a queue, but an attribute selector does not apply to that type of queue, the specific value MQIAV_NOT_APPLICABLE is returned for the corresponding element in the *IntAttrs* array.

If the *IntAttrCount* or *SelectorCount* parameter is zero, *IntAttrs* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler might be null.

CharAttrLength (MQLONG) – input

This is the length in bytes of the *CharAttrs* parameter.

This must be at least the sum of the lengths of the requested character attributes (see *Selectors*). Zero is a valid value.

CharAttrs (MQCHARxCharAttrLength) – output

This is the buffer in which the character attributes are returned, concatenated together. The length of the buffer is given by the *CharAttrLength* parameter.

Character attributes are returned in the same order as the MQCA_* selectors in the *Selectors* parameter. The length of each attribute string is fixed for each attribute (see *Selectors*), and the value in it is padded to the right with blanks if necessary. If the buffer is larger than that needed to contain all 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 does not apply to that type of queue, a character string consisting entirely of asterisks (*) is returned as the value of that attribute in *CharAttrs*.

If the *CharAttrLength* or *SelectorCount* parameter is zero, *CharAttrs* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler might be null.

CompCode (MQLONG) – output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_CHAR_ATTRS_TOO_SHORT
(2008, X'7D8') Not enough space allowed for character attributes.

MQRC_INT_ATTR_COUNT_TOO_SMALL
(2022, X'7E6') Not enough space allowed for integer attributes.

MQRC_SELECTOR_NOT_FOR_TYPE
(2068, X'814') Selector not applicable to queue type.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_STRUC_FAILED
(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CHAR_ATTR_LENGTH_ERROR
(2006, X'7D6') Length of character attributes not valid.

MQRC_CHAR_ATTRS_ERROR
(2007, X'7D7') Character attributes string not valid.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_INT_ATTR_COUNT_ERROR
(2021, X'7E5') Count of integer attributes not valid.

MQRC_INT_ATTRS_ARRAY_ERROR
(2023, X'7E7') Integer attributes array not valid.

MQRC_NOT_OPEN_FOR_INQUIRE
(2038, X'7F6') Queue not open for inquire.

MQRC_OBJECT_CHANGED
(2041, X'7F9') Object definition changed since opened.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_Q_DELETED
(2052, X'804') Queue has been deleted.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SELECTOR_COUNT_ERROR
(2065, X'811') Count of selectors not valid.

MQRC_SELECTOR_ERROR
(2067, X'813') Attribute selector not valid.

MQRC_SELECTOR_LIMIT_EXCEEDED
(2066, X'812') Count of selectors too big.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information on these codes, see:

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

Usage notes for MQINQ

Consider the following points when you intend to use the MQINQ call.

1. The values returned are a snapshot of the selected attributes. There is no guarantee that the attributes will not change before the application can act upon the returned values.
2. When you open a model queue, a dynamic local queue is created. This is true even if you open the model queue to inquire about its attributes.

The attributes of the dynamic queue (with certain exceptions) are the same as those of the model queue at the time that the dynamic queue is created. If you subsequently use the MQINQ call on this queue, the queue manager returns the attributes of the dynamic queue, and not those of the model queue. See Table 96 on page 617 for details of which attributes of the model queue are inherited by the dynamic queue.
3. If the object being inquired is an alias queue, the attribute values returned by the MQINQ call are those of the alias queue, and not those of the base queue or topic to which the alias resolves.
4. If the object being inquired is a cluster queue, the attributes that can be inquired depend on how the queue is opened:
 - If the cluster queue is opened for inquire plus one or more of input, browse, or set, there must be a local instance of the cluster queue for the open to succeed. In this case the attributes that can be inquired are those valid for local queues.
 - If the cluster queue is opened for inquire alone, or inquire and output, only the attributes listed below can be inquired; the *QType* attribute has the value MQQT_CLUSTER in this case:
 - MQCA_Q_DESC
 - MQCA_Q_NAME
 - MQIA_DEF_BIND
 - MQIA_DEF_PERSISTENCE
 - MQIA_DEF_PRIORITY
 - MQIA_INHIBIT_PUT
 - MQIA_Q_TYPE

If the cluster queue is opened with no fixed binding (that is, MQOO_BIND_NOT_FIXED specified on the MQOPEN call, or MQOO_BIND_AS_Q_DEF specified when the *DefBind* attribute has the value MQBND_BIND_NOT_FIXED), successive MQINQ calls for the queue might inquire different instances of the cluster queue, although usually all the instances have the same attribute values.

- An alias queue object can be defined for a cluster. Since TARGTYPE and TARGET are not cluster attributes, the process performing an MQOPEN process on the alias queue is not aware of the object to which the alias resolves.

During the initial MQOPEN, the alias queue simply resolves to a queue manager and a queue in the cluster. Name resolution takes place again at the remote queue manager and it is here that the TARGTYPE of the alias queue is resolved.

If the alias queue resolves to a topic alias, then publication of messages put to the alias queue takes place at this remote queue manager.

For more information about cluster queues, refer to *WebSphere MQ Queue Manager Clusters*.

5. If you want to inquire a number of attributes, and subsequently set some of them using the MQSET call, you might want to position the attributes to be set at the beginning of the selector arrays, so that the same arrays (with reduced counts) can be used for MQSET.
6. If more than one of the warning situations arise (see the *CompCode* parameter), the reason code returned is the *first* one in the following list that applies:
 - a. MQRC_SELECTOR_NOT_FOR_TYPE
 - b. MQRC_INT_ATTR_COUNT_TOO_SMALL
 - c. MQRC_CHAR_ATTRS_TOO_SHORT
7. For more information about object attributes, see:
 - “Attributes for queues” on page 615
 - “Attributes for namelists” on page 651
 - “Attributes for process definitions” on page 654
 - “Attributes for the queue manager” on page 575

Language invocations for MQINQ

The MQINQ call is supported in the programming languages shown below.

C invocation

```
MQINQ (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,
      CharAttrLength, CharAttrs, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;          /* Connection handle */
MQHOBJ  Hobj;           /* Object handle */
MQLONG  SelectorCount; /* Count of selectors */
MQLONG  Selectors[n];  /* Array of attribute selectors */
MQLONG  IntAttrCount;  /* Count of integer attributes */
MQLONG  IntAttrs[n];   /* Array of integer attributes */
MQLONG  CharAttrLength; /* Length of character attributes buffer */
MQCHAR  CharAttrs[n];  /* Character attributes */
MQLONG  CompCode;      /* Completion code */
MQLONG  Reason;        /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQINQ' USING HCONN, HOBJ, SELECTORCOUNT, SELECTORS-TABLE,
                  INTATTRCOUNT, INTATTRS-TABLE, CHARATTRLENGTH,
                  CHARATTRS, COMPCODE, REASON.
```

Declare the parameters as follows:

```

** Connection handle
01 HCONN          PIC S9(9) BINARY.
** Object handle
01 HOBJ          PIC S9(9) BINARY.
** Count of selectors
01 SELECTORCOUNT PIC S9(9) BINARY.
** Array of attribute selectors
01 SELECTORS-TABLE.
02 SELECTORS      PIC S9(9) BINARY OCCURS n TIMES.
** Count of integer attributes
01 INTATTRCOUNT PIC S9(9) BINARY.
** Array of integer attributes
01 INTATTRS-TABLE.
02 INTATTRS      PIC S9(9) BINARY OCCURS n TIMES.
** Length of character attributes buffer
01 CHARATTRLENGTH PIC S9(9) BINARY.
** Character attributes
01 CHARATTRS      PIC X(n).
** Completion code
01 COMPCODE       PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON         PIC S9(9) BINARY.

```

PL/I invocation

```

call MQINQ (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount,
           IntAttrs, CharAttrLength, CharAttrs, CompCode, Reason);

```

Declare the parameters as follows:

```

dcl Hconn          fixed bin(31); /* Connection handle */
dcl Hobj           fixed bin(31); /* Object handle */
dcl SelectorCount  fixed bin(31); /* Count of selectors */
dcl Selectors(n)   fixed bin(31); /* Array of attribute selectors */
dcl IntAttrCount   fixed bin(31); /* Count of integer attributes */
dcl IntAttrs(n)    fixed bin(31); /* Array of integer attributes */
dcl CharAttrLength fixed bin(31); /* Length of character attributes
                                buffer */
dcl CharAttrs      char(n);       /* Character attributes */
dcl CompCode       fixed bin(31); /* Completion code */
dcl Reason         fixed bin(31); /* Reason code qualifying
                                CompCode */

```

System/390 assembler invocation

```

CALL MQINQ,(HCONN,HOBJ,SELECTORCOUNT,SELECTORS,INTATTRCOUNT, X
           INTATTRS,CHARATTRLENGTH,CHARATTRS,COMPCODE,REASON)

```

Declare the parameters as follows:

```

HCONN      DS F      Connection handle
HOBJ       DS F      Object handle
SELECTORCOUNT DS F      Count of selectors
SELECTORS  DS (n)F   Array of attribute selectors
INTATTRCOUNT DS F      Count of integer attributes
INTATTRS   DS (n)F   Array of integer attributes
CHARATTRLENGTH DS F      Length of character attributes buffer
CHARATTRS  DS CL(n)  Character attributes
COMPCODE   DS F      Completion code
REASON     DS F      Reason code qualifying COMPCODE

```

Visual Basic invocation

```

MQINQ Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,
      CharAttrLength, CharAttrs, CompCode, Reason

```

Declare the parameters as follows:

Dim Hconn	As Long	'Connection handle'
Dim Hobj	As Long	'Object handle'
Dim SelectorCount	As Long	'Count of selectors'
Dim Selectors	As Long	'Array of attribute selectors'
Dim IntAttrCount	As Long	'Count of integer attributes'
Dim IntAttrs	As Long	'Array of integer attributes'
Dim CharAttrLength	As Long	'Length of character attributes buffer'
Dim CharAttrs	As String	'Character attributes'
Dim CompCode	As Long	'Completion code'
Dim Reason	As Long	'Reason code qualifying CompCode'

MQINQMP - Inquire message property

The MQINQMP call returns the value of a property of a message.

Syntax for MQINQMP

MQINQMP (*Hconn, Hmsg, InqPropOps, Name, PropDesc, Type, ValueLength, Value, DataLength, CompCode, Reason*)

Parameters for MQINQMP

The MQINQMP call has the following parameters.

Hconn (MQHCONN) - Input

This handle represents the connection to the queue manager. The value of *Hconn* must match the connection handle that was used to create the message handle specified in the *Hmsg* parameter.

If the message handle was created using MQHC_UNASSOCIATED_HCONN then a valid connection must be established on the thread inquiring a property of the message handle otherwise the call fails with MQRC_CONNECTION_BROKEN.

Hmsg (MQHMSG) - input

This is the message handle to be inquired. The value was returned by a previous MQCRTMH call.

InqPropOps (MQIMPO) - Input

See the MQIMPO data type for details.

Name (MQCHARV) - input

The name of the property to inquire.

If no property with this name can be found, the call fails with reason MQRC_PROPERTY_NOT_AVAILABLE.

You can use the wildcard character '%' at the end of the property name. The wildcard matches zero or more characters, including the '.' character. This allows an application to inquire the value of many properties. Call MQINQMP with option MQIMPO_INQ_FIRST to get the first matching property and again with the option MQIMPO_INQ_NEXT to get the next matching property. When no more matching properties are available, the call fails with

MQRC_PROPERTY_NOT_AVAILABLE . If the *ReturnedName* field of the InqPropOpts structure is initialized with an address or offset for the returned name of the property, this is filled in on return from MQINQMP with the same of the property that has been matched. If the *VSBufSize* field of the *ReturnedName* in the InqPropOpts structure is less than the length of the returned property name the completion code is set MQCC_FAILED with reason MQRC_PROPERTY_NAME_TOO_BIG.

Properties that have known synonyms are returned as follows:

1. • Properties with the prefix "mqps." are returned with the MQ property name e.g. "MQTopicString" is the returned name rather than "mqps.Top"
2. Properties with the prefix "jms." or "mcd." are returned as the JMS header field name, for example, "JMSExpiration" is the returned name rather than "jms.Exp".
3. Properties with the prefix "usr." are returned without that prefix, for example, "Color" is returned rather than "usr.Color".

Properties with synonyms are only returned once.

In the C programming language, the following macro variables are defined for inquiring on all properties and all properties that begin 'usr', respectively:

MQPROP_INQUIRE_ALL

Inquire on all properties of the message.

MQPRP_INQUIRE_ALL_USR

Inquire on all properties of the message that start 'usr.'. The returned name is returned without the 'usr.' prefix.

If MQIMP_INQ_NEXT is specified but Name has changed since the previous call or this is the first call, then MQIMPO_INQ_FIRST is implied.

See the *WebSphere MQ Application Programming Guide* for further information about the use of property names.

PropDesc (MQPD) - output

This structure is used to define the attributes of a property, including what happens if the property is not supported, what message context the property belongs to, and what messages the property should be copied into. See MQPD for details of this structure.

Type (MQLONG) - input/output

On return from the MQINQMP call this parameter is set to the data type of *Value*. The data type can be any of the following:

MQTYPE_BOOLEAN

A boolean.

MQTYPE_BYTE_STRING

a byte string.

MQTYPE_INT8

An 8-bit signed integer.

MQTYPE_INT16

A 16-bit signed integer.

MQTYPE_INT32
A 32-bit signed integer.

MQTYPE_INT64
A 64-bit signed integer.

MQTYPE_FLOAT32
A 32-bit floating-point number.

MQTYPE_FLOAT64
A 64-bit floating-point number.

MQTYPE_STRING
A character string.

MQTYPE_NULL
The property exists but has a null value.

If the data type of the property value is not recognized then `MQTYPE_STRING` is returned and a string representation of the value is placed into the *Value* area. A string representation of the data type can be found in the *TypeString* field of the *InqPropOpts* parameter. A warning completion code is returned with reason `MQRC_PROP_TYPE_NOT_SUPPORTED`.

Additionally, if the option `MQIMPO_CONVERT_TYPE` is specified, conversion of the property value is requested. Use *Type* as an input to specify the data type that you want the property to be returned as. See the description of the `MQIMPO_CONVERT_TYPE` option of the `MQIMPO` structure for details of data type conversion.

If you do not request type conversion, you can use the following value on input:

MQTYPE_AS_SET
The value of the property is returned without converting its data type.

ValueLength (MQLONG) - input

The length in bytes of the *Value* area. Specify zero for properties that you do not require the value returned for. These could be properties which are designed by an application to have a null value or an empty string. Also specify zero if the `MQIMPO_QUERY_LENGTH` option has been specified; in this case no value is returned.

Value (MQBYTExValueLength) - output

This is the area to contain the inquired property value. The buffer should be aligned on a boundary appropriate for the value being returned. Failure to do so may result in an error when the value is later accessed.

If *ValueLength* is less than the length of the property value, as much of the property value as possible is moved into *Value* and the call fails with completion code `MQCC_FAILED` and reason `MQRC_PROPERTY_VALUE_TOO_BIG`.

The character set of the data in *Value* is given by the `ReturnedCCSID` field in the *InqPropOpts* parameter. The encoding of the data in *Value* is given by the `ReturnedEncoding` field in the *InqPropOpts* parameter.

In the C programming language, the parameter is declared as a pointer-to-void; the address of any type of data can be specified as the parameter.

If the *ValueLength* parameter is zero, *Value* is not referred to and its value passed by programs written in C or System/390 assembler can be null.

DataLength (MQLONG) - output

This is the length in bytes of the actual property value as returned in the *Value* area.

If *DataLength* is less than the property value length, *DataLength* is still filled in on return from the MQINQMP call. This allows the application to determine the size of the buffer required to accommodate the property value, and then reissue the call with a buffer of the appropriate size.

The following values may also be returned.

If the *Type* parameter is set to MQTYPE_STRING or MQTYPE_BYTE_STRING:

MQVL_EMPTY_STRING

The property exists but contains no characters or bytes.

CompCode (MQLONG) – output

The completion code; it is one of the following:

MQCC_OK

Successful completion.

MQCC_WARNING

Warning (partial completion).

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_PROP_NAME_NOT_CONVERTED

(2492, X'09BC') Returned property name not converted.

MQRC_PROP_VALUE_NOT_CONVERTED

(2466, X'09A2') Property value not converted.

MQRC_PROP_TYPE_NOT_SUPPORTED

(2467, X'09A3') Property data type is not supported.

MQRC_RFH_FORMAT_ERROR

(2421, X'0975') An MQRFH2 folder containing properties could not be parsed.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE

(2204, X'089C') Adapter not available.

| **MQRC_ADAPTER_SERV_LOAD_ERROR**
 | (2130, X'0852') Unable to load adapter service module.

| **MQRC_ASID_MISMATCH**
 | (2157, X'086D') Primary and home ASIDs differ.

| **MQRC_BUFFER_ERROR**
 | (2004, X'07D4') Value parameter not valid.

| **MQRC_BUFFER_LENGTH_ERROR**
 | (2005, X'07D5') Value length parameter not valid.

| **MQRC_CALL_IN_PROGRESS**
 | (2219, X'08AB') MQI call entered before previous call completed.

| **MQRC_CONNECTION_BROKEN**
 | (2009, X'07D9') Connection to queue manager lost.

| **MQRC_DATA_LENGTH_ERROR**
 | (2010, X'07DA') Data length parameter not valid.

| **MQRC_IMPO_ERROR**
 | (2464, X'09A0') Inquire message property options structure not valid.

| **MQRC_HMSG_ERROR**
 | (2460, X'099C') Message handle not valid.

| **MQRC_MSG_HANDLE_IN_USE**
 | (2499, X'09C3') Message handle already in use.

| **MQRC_OPTIONS_ERROR**
 | (2046, X'07F8') Options not valid or not consistent.

| **MQRC_PD_ERROR**
 | (2482, X'09B2') Property descriptor structure not valid.

| **MQRC_PROP_CONV_NOT_SUPPORTED**
 | (2470, X'09A6') Conversion from the actual to requested data type not
 | supported.

| **MQRC_PROPERTY_NAME_ERROR**
 | (2442, X'098A') Invalid property name.

| **MQRC_PROPERTY_NAME_TOO_BIG**
 | (2465, X'09A1') Property name too big for returned name buffer.

| **MQRC_PROPERTY_NOT_AVAILABLE**
 | (2471, X'09A7) Property not available.

| **MQRC_PROPERTY_VALUE_TOO_BIG**
 | (2469, X'09A5') Property value too big for the Value area.

| **MQRC_PROP_NUMBER_FORMAT_ERROR**
 | (2472, X'09A8') Number format error encountered in value data.

| **MQRC_PROPERTY_TYPE_ERROR**
 | (2473, X'09A9') Invalid requested property type.

| **MQRC_SOURCE_CCSID_ERROR**
 | (2111, X'083F') Property name coded character set identifier not valid.

| **MQRC_STORAGE_NOT_AVAILABLE**
 | (2071, X'0871') Insufficient storage available.

| **MQRC_UNEXPECTED_ERROR**
 | (2195, X'0893') Unexpected error occurred.

For detailed information on these codes, see:

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

Language invocations for MQINQMP

C invocation

```
MQINQMP (Hconn, Hmsg, &InqPropOpts, &Name, &PropDesc, &Type,  
ValueLength, Value, &DataLength, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;      /* Connection handle */  
MQHMSG Hmsg;        /* Message handle */  
MQDIMPO InqPropOpts; /* Options that control the action of MQINQMP */  
MQCHARV Name;       /* Property name */  
MQPD PropDesc;      /* Property descriptor */  
MQLONG Type;        /* Property data type */  
MQLONG ValueLength; /* Length in bytes of the Value area */  
MQBYTE Value[n];    /* Area to contain the property value */  
MQLONG DataLength;  /* Length of the property value */  
MQLONG CompCode;    /* Completion code */  
MQLONG Reason;      /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQINQMP' USING HCONN, HMSG, INQMSGOPTS, NAME, PROPDESC, TYPE,  
VALUELENGTH, VALUE, DATALENGTH, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle  
01 HCONN PIC S9(9) BINARY.  
** Message handle  
01 HMSG PIC S9(18) BINARY.  
** Options that control the action of MQINQMP  
01 INQMSGOPTS.  
COPY CMQIMPOV.  
** Property name  
01 NAME.  
COPY CMQCHRNV.  
** Property descriptor  
01 PROPDESC.  
COPY CMQPDV.  
** Property data type  
01 TYPE PIC S9(9) BINARY.  
** Length in bytes of the VALUE area  
01 VALUELENGTH PIC S9(9) BINARY.  
** Area to contain the property value  
01 VALUE PIC X(n).  
** Length of the property value  
01 DATALENGTH PIC S9(9) BINARY.  
** Completion code  
01 COMPCODE PIC S9(9) BINARY.  
** Reason code qualifying COMPCODE  
01 REASON PIC S9(9) BINARY.
```

PL/I invocation

```
call MQINQMP (Hconn, Hmsg, InqPropOpts, Name, PropDesc, Type,  
ValueLength, Value, DataLength, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn fixed bin(31); /* Connection handle */  
dcl Hmsg fixed bin(63); /* Message handle */  
dcl InqPropOpts like MQIMPO; /* Options that control the action of MQINQMP */
```

```

dc1 Name          like MQCHARV; /* Property name */
dc1 PropDesc      like MQPD; /* Property descriptor */
dc1 Type          fixed bin (31); /* Property data type */
dc1 ValueLength   fixed bin (31); /* Length in bytes of the Value area */
dc1 Value         char (n); /* Area to contain the property value */
dc1 DataLength    fixed bin (31); /* Length of the property value */
dc1 CompCode      fixed bin (31); /* Completion code */
dc1 Reason        fixed bin (31); /* Reason code qualifying CompCode */

```

System/390 assembler invocation

Parameters used for the System/390 assembler invocation of MQINQMP.

```

CALL MQINQMP, (HCONN,HMSG,INQMSGOPTS,NAME,PROPDESC,TYPE,
VALUELENGTH,VALUE,DATALength,COMPCODE,REASON)

```

Declare the parameters as follows:

HCONN	DS	F	Connection handle
HMSG	DS	D	Message handle
INQMSGOPTS	CMQIMPOA	,	Options that control the action of MQINQMP
NAME	CMQCHRVA	,	Property name
PROPDESC	CMQPDA	,	Property descriptor
TYPE	DS	F	Property data type
VALUELENGTH	DS	F	Length in bytes of the VALUE area
VALUE	DS	CL(n)	Area to contain the property value
DATALength	DS	F	Length of the property value
COMPCODE	DS	F	Completion code
REASON	DS	F	Reason code qualifying COMPCODE

MQMHBUF - Convert message handle into buffer

The MQMHBUF converts a message handle into a buffer and is the inverse of the MQBUFMH call.

Syntax for MQMHBUF

```

MQMHBUF (Hconn, Hmsg, MsgHBufOpts, Name, MsgDesc, BufferLength, Buffer, DataLength, CompCode, Reason)

```

Parameters for MQMHBUF

The MQMHBUF call has the following parameters.

Hconn (MQHCONN) – input

This handle represents the connection to the queue manager. The value of *Hconn* must match the connection handle that was used to create the message handle specified in the *Hmsg* parameter.

If the message handle was created using MQHC_UNASSOCIATED_HCONN, a valid connection must be established on the thread deleting the message handle. If a valid connection is not established, the call fails with MQRC_CONNECTION_BROKEN.

Hmsg (MQHMSG) – input

This is the message handle for which a buffer is required.

The value was returned by a previous MQCRTMH call.

MsgHBufOpts (MQMHBO) – input

The MQMHBO structure allows applications to specify options that control how buffers are produced from message handles.

See “MQMHBO – Message handle to buffer options” on page 247 for details.

Name (MQCHARV) - input

The name of the property or properties to put into the buffer.

If no property matching the name can be found, the call fails with MQRC_PROPERTY_NOT_AVAILABLE.

Wildcards

You can use a wildcard to put more than one property into the buffer. To do this, use the wildcard character ‘%’ at the end of the property name. This wildcard matches zero or more characters, including the ‘.’ character.

In the C programming language, the following macro variables are defined for inquiring on all properties and all properties that begin ‘usr’, respectively:

MQPROP_INQUIRE_ALL

Put all properties of the message into the buffer

MQPROP_INQUIRE_ALL_USR

Put all properties of the message that start with the characters ‘usr.’ into the buffer.

See the *WebSphere MQ Application Programming Guide* for further information about the use of property names.

MsgDesc (MQMD) – input/output

The *MsgDesc* structure describes the contents of the buffer area.

On output, the *Encoding*, *CodedCharSetId* and *Format* fields are set to correctly describe the encoding, character set identifier, and format of the data in the buffer area as written by the call.

Data in this structure is in the character set and encoding of the application.

BufferLength (MQLONG) - input

BufferLength is the length of the Buffer area, in bytes.

Buffer (MQBYTEExBufferLength) - output

Buffer defines the area to contain the message properties. You should align the buffer on a 4-byte boundary.

If *BufferLength* is less than the length required to store the properties in *Buffer*, MQMHBUF fails with MQRC_PROPERTY_VALUE_TOO_BIG.

The contents of the buffer can change even if the call fails.

DataLength (MQLONG) - output

DataLength is the length, in bytes, of the returned properties in the buffer. If the value is zero, no properties matched the value given in *Name* and the call fails with reason code MQRC_PROPERTY_NOT_AVAILABLE.

If *BufferLength* is less than the length required to store the properties in the buffer, the MQMHBUFF call fails with MQRC_PROPERTY_VALUE_TOO_BIG, but a value is still entered into *DataLength*. This allows the application to determine the size of the buffer required to accommodate the properties, and then reissue the call with the required *BufferLength*.

CompCode (MQLONG) – output

The completion code; it is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE

(2204, X'089C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR

(2130, X'852') Unable to load adapter service module.

MQRC_ASID_MISMATCH

(2157, X'86D') Primary and home ASIDs differ.

MQRC_MHBO_ERROR

(2501, X'095C') Message handle to buffer options structure not valid.

MQRC_BUFFER_ERROR

(2004, X'07D4') Buffer parameter not valid.

MQRC_BUFFER_LENGTH_ERROR

(2005, X'07D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS

(2219, X'08AB') MQI call entered before previous call completed.

MQRC_CONNECTION_BROKEN

(2009, X'07D9') Connection to queue manager lost.

MQRC_DATA_LENGTH_ERROR

(2010, X'07DA') Data length parameter not valid.

MQRC_HMSG_ERROR

(2460, X'099C') Message handle not valid.

MQRC_MD_ERROR
(2026, X'07EA') Message descriptor not valid.

MQRC_MSG_HANDLE_IN_USE
(2499, X'09C3') Message handle already in use.

MQRC_OPTIONS_ERROR
(2046, X'07FE') Options not valid or not consistent.

MQRC_PROPERTY_NAME_ERROR
(2442, X'098A') Property name is not valid.

MQRC_PROPERTY_NOT_AVAILABLE
(2471, X'09A7') Property not available.

MQRC_PROPERTY_VALUE_TOO_BIG
(2469, X'09A5') BufferLength value is too small to contain specified properties.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

Usage notes for MQMHBUF

MQMHBUF converts a message handle into a buffer.

You can use it with an MQGET API exit to access certain properties, using the message property APIs, and then pass these in a buffer back to an application designed to use MQRFH2 headers rather than message handles.

This call is the inverse of the MQBUFMH call, which you can use to parse message properties from a buffer into a message handle.

Language invocations for MQMHBUF

The MQMHBUF call is supported in the programming languages shown below.

C invocation

```
MQMHBUF (Hconn, Hmsg, &MsgHBufOpts, &Name, &MsgDesc, BufferLength, Buffer,
        &DataLength, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;      /* Connection handle */
MQHMSG  Hmsg;       /* Message handle */
MQMHBO  MsgHBufOpts; /* Options that control the action of MQMHBUF */
MQCHARV Name;      /* Property name */
MQMD    MsgDesc;   /* Message descriptor */
MQLONG  BufferLength; /* Length in bytes of the Buffer area */
MQBYTE  Buffer[n];  /* Area to contain the properties */
MQLONG  DataLength; /* Length of the properties */
MQLONG  CompCode;  /* Completion code */
MQLONG  Reason;    /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQMHBUF' USING HCONN, HMSG, MSGHBUFOPTS, NAME, MSGDESC,
                   BUFFERLENGTH, BUFFER, DATALENGTH, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN          PIC S9(9) BINARY.
** Message handle
01 HMSG           PIC S9(18) BINARY.
```

```

** Options that control the action of MQMHBUF
01 MSGHBUFOPTS.
   COPY CMQMHBVOV.
** Property name
01 NAME
   COPY CMQCHRVV.
** Message descriptor
01 MSGDESC
   COPY CMQMDV.
** Length in bytes of the Buffer area */
01 BUFFERLENGTH PIC S9(9) BINARY.
** Area to contain the properties
01 BUFFER      PIC X(n).
** Length of the properties
01 DATALENGTH PIC S9(9) BINARY.
** Completion code
01 COMPCODE    PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON      PIC S9(9) BINARY.

```

PL/I invocation

```

call MQMHBUF (Hconn, Hmsg, MsgHBufOpts, Name, MsgDesc, BufferLength, Buffer,
              DataLength, CompCode, Reason);

```

Declare the parameters as follows:

```

dc1 Hconn      fixed bin(31); /* Connection handle */
dc1 Hmsg       fixed bin(63); /* Message handle */
dc1 MsgHBufOpts like MQMHB0; /* Options that control the action of MQMHBUF */
dc1 Name       like MQCHARV; /* Property name */
dc1 MsgDesc    like MQMD; /* Message descriptor */
dc1 BufferLength fixed bin(31); /* Length in bytes of the Buffer area */
dc1 Buffer      char(n); /* Area to contain the properties */
dc1 DataLength fixed bin(31); /* Length of the properties */
dc1 CompCode   fixed bin(31); /* Completion code */
dc1 Reason     fixed bin(31); /* Reason code qualifying CompCode */

```

System/390 assembler invocation

```

CALL MQMHBUF,(HCONN,HMSG,MSGHBUFOPTS,NAME,MSGDESC,BUFFERLENGTH,
              BUFFER,DATALENGTH,COMPCODE,REASON)

```

Declare the parameters as follows:

HCONN	DS	F	Connection handle
HMSG	DS	D	Message handle
MSGHBUFOPTS	CMQMHB0A	,	Options that control the action of MQMHBUF
NAME	CMQCHRVA	,	Property name
MSGDESC	CMQMDA	,	Message descriptor
BUFFERLENGTH	DS	F	Length in bytes of the BUFFER area
BUFFER	DS	CL(n)	Area to contain the properties
DATALENGTH	DS	F	Length of the properties
COMPCODE	DS	F	Completion code
REASON	DS	F	Reason code qualifying COMPCODE

MQOPEN – Open object

The MQOPEN call establishes access to an object.

The following types of object are valid:

- Queue (including distribution lists)
- Namelist
- Process definition
- Queue manager

- Topic

Syntax for MQOPEN

MQOPEN (*Hconn*, *ObjDesc*, *Options*, *Hobj*, *CompCode*, *Reason*)

Parameters for MQOPEN

The MQOPEN call has the following parameters.

Hconn (MQHCONN) – input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on i5/OS for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN

Default connection handle.

ObjDesc (MQOD) – input/output

This is a structure that identifies the object to be opened; see “MQOD – Object descriptor” on page 249 for details.

If the *ObjectName* field in the *ObjDesc* parameter is the name of a model queue, a dynamic local queue is created with the attributes of the model queue; this happens whatever options you specify on the *Options* parameter. Subsequent operations using the *Hobj* returned by the MQOPEN call are performed on the new dynamic queue, and not on the model queue. This is true even for the MQINQ and MQSET calls. The name of the model queue in the *ObjDesc* parameter is replaced with the name of the dynamic queue created. The type of the dynamic queue is determined by the value of the *DefinitionType* attribute of the model queue (see “Attributes for queues” on page 615). For information about the close options applicable to dynamic queues, see the description of the MQCLOSE call.

Options (MQLONG) – input

You must specify at least one of the following options:

- MQOO_BROWSE
- MQOO_INPUT_* (only one of these)
- MQOO_INQUIRE
- MQOO_OUTPUT
- MQOO_SET

See below for details of these options; other options can be specified as required. If more than one option is required, the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

Combinations that are not valid are noted; all other combinations are valid. Only options that are applicable to the type of object specified by *ObjDesc* are allowed. The following table shows valid MQOPEN options for queries and topics.

Option	Alias (note 1)	Local and Model	Remote	Nonlocal Cluster	Distribution list	Topic
MQOO_INPUT_AS_Q_DEF	Yes	Yes	No	No	No	No
MQOO_INPUT_SHARED	Yes	Yes	No	No	No	No
MQOO_INPUT_EXCLUSIVE	Yes	Yes	No	No	No	No
MQOO_BROWSE	Yes	Yes	No	No	No	No
MQOO_OUTPUT	Yes	Yes	Yes	Yes	Yes	Yes
MQOO_INQUIRE	Yes	Yes	Note 2	Yes	No	No
MQOO_SET	Yes	Yes	Note 2	No	No	No
MQOO_BIND_ON_OPEN (note 3)	Yes	Yes	Yes	Yes	Yes	No
MQOO_BIND_NOT_FIXED (note 3)	Yes	Yes	Yes	Yes	Yes	No
MQOO_BIND_AS_Q_DEF (note 3)	Yes	Yes	Yes	Yes	Yes	No
MQOO_SAVE_ALL_CONTEXT	Yes	Yes	No	No	No	No
MQOO_PASS_IDENTITY_CONTEXT	Yes	Yes	Yes	Yes	Yes	Note 4
MQOO_PASS_ALL_CONTEXT	Yes	Yes	Yes	Yes	Yes	Yes
MQOO_SET_IDENTITY_CONTEXT	Yes	Yes	Yes	Yes	Yes	Note 4
MQOO_SET_ALL_CONTEXT	Yes	Yes	Yes	Yes	Yes	Yes
MQOO_ALTERNATE_USER_AUTHORITY	Yes	Yes	Yes	Yes	Yes	Yes
MQOO_FAIL_IF QUIESCING	Yes	Yes	Yes	Yes	Yes	Yes
MQOO_RESOLVE_LOCAL_Q	Yes	Yes	Yes	Yes	No	No

Notes:

1. The validity of options for aliases depends on the validity of the option for the queue to which the alias resolves.
2. This option is valid only for the local definition of a remote queue.
3. This option can be specified for any queue type, but is ignored if the queue is not a cluster queue.
4. These attributes can be used with a topic, but affect only the context set for the retained message, not the context fields sent to any subscriber.

Access options: The following options control the type of operations that can be performed on the object:

MQOO_INPUT_AS_Q_DEF

Open queue to get messages using queue-defined default.

The queue is opened for use with subsequent MQGET calls. The type of access is either shared or exclusive, depending on the value of the *DefInputOpenOption* queue attribute; see “Attributes for queues” on page 615 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.

MQOO_INPUT_SHARED

Open queue to get messages with shared access.

The queue is opened for use with subsequent MQGET calls. The call can succeed if the queue is currently open by this or another application with MQOO_INPUT_SHARED, but fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open with MQOO_INPUT_EXCLUSIVE.

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues.

MQOO_INPUT_EXCLUSIVE

Open queue to get messages with exclusive access.

The queue is opened for use with subsequent MQGET calls. The call fails with reason code MQRRC_OBJECT_IN_USE if the queue is currently open by this or another application for input of any type (MQOO_INPUT_SHARED or MQOO_INPUT_EXCLUSIVE).

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues.

MQOO_OUTPUT

Open queue to put messages, or a topic or topic string to publish messages.

The queue or topic is opened for use with subsequent MQPUT calls.

An MQOPEN call with this option can succeed even if the *InhibitPut* queue attribute is set to MQQA_PUT_INHIBITED (although subsequent MQPUT calls will fail while the attribute is set to this value).

This option is valid for all types of queue, including distribution lists, and topics.

The following notes apply to these options:

- Only one of these options can be specified.
- An MQOPEN call with one of these options can succeed even if the *InhibitGet* queue attribute is set to MQQA_GET_INHIBITED (although subsequent MQGET calls will fail while the attribute is set to this value).
- If the queue is defined as not being shareable (that is, the *Shareability* queue attribute has the value MQQA_NOT_SHAREABLE), attempts to open the queue for shared access are treated as attempts to open the queue with exclusive access.
- If an alias queue is opened with one of these options, the test for exclusive use (or for whether another application has exclusive use) is against the base queue to which the alias resolves.
- These options are not valid if *ObjectQMgrName* is the name of a queue manager alias; this is true even if the value of the *RemoteQMgrName* attribute in the local definition of a remote queue used for queue-manager aliasing is the name of the local queue manager.

MQOO_BROWSE

Open queue to browse messages.

The queue is opened for use with subsequent MQGET calls with one of the following options:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_NEXT
- MQGMO_BROWSE_MSG_UNDER_CURSOR

This is allowed even if the queue is currently open for MQOO_INPUT_EXCLUSIVE. An MQOPEN call with the MQOO_BROWSE option establishes a browse cursor, and positions it logically before the first message on the queue; see MQGMO - Options field 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 that are not queues. It is also not valid if *ObjectQMgrName* is the name of a queue manager alias; this is true even if the value of the *RemoteQMgrName* attribute in the local definition of a remote queue used for queue-manager aliasing is the name of the local queue manager.

MQOO_CO_OP

Open as a cooperating member of the set of handles.

This option is valid only with the MQOO_BROWSE option. If it is specified without MQOO_BROWSE, MQOPEN returns with MQRC_OPTIONS_ERROR.

The handle returned is considered to be a member of a cooperating set of handles for subsequent MQGET calls with one of the following options:

- MQGMO_MARK_BROWSE_CO_OP
- MQGMO_UNMARKED_BROWSE_MSG
- MQGMO_UNMARK_BROWSE_CO_OP

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues.

MQOO_OUTPUT

Open queue to put messages.

The queue is opened for use with subsequent MQPUT calls.

An MQOPEN call with this option can succeed even if the *InhibitPut* queue attribute is set to MQQA_PUT_INHIBITED (although subsequent MQPUT calls will fail while the attribute is set to this value).

This option is valid for all types of queue, including distribution lists.

MQOO_INQUIRE

Open object to inquire attributes.

The queue, namelist, process definition, or queue manager is opened for use with subsequent MQINQ calls.

This option is valid for all types of object other than distribution lists. It is not valid if *ObjectQMgrName* is the name of a queue manager alias; this is true even if the value of the *RemoteQMgrName* attribute in the local definition of a remote queue used for queue-manager aliasing is the name of the local queue manager.

MQOO_SET

Open queue to set attributes.

The queue is opened for use with subsequent MQSET calls.

This option is valid for all types of queue other than distribution lists. It is not valid if *ObjectQMgrName* is the name of a local definition of a remote queue; this is true even if the value of the *RemoteQMgrName* attribute in the local definition of a remote queue used for queue-manager aliasing is the name of the local queue manager.

Binding options: The following options apply when the object being opened is a cluster queue; these options control the binding of the queue handle to a particular instance of the cluster queue:

MQOO_BIND_ON_OPEN

The local queue manager binds the queue handle to a particular instance of

the destination queue when the queue is opened. As a result, all messages put using this handle are sent to the same instance of the destination queue, and by the same route.

This option is valid only for queues, and affects only cluster queues. If specified for a queue that is not a cluster queue, the option is ignored.

MQOO_BIND_NOT_FIXED

This stops the local queue manager binding the queue handle to a particular instance of the destination queue. As a result, successive MQPUT calls using this handle send the messages to *different* instances of the destination queue, or to the same instance but by different routes. It also allows the instance selected to be changed subsequently by the local queue manager, by a remote queue manager, or by a message channel agent (MCA), according to network conditions.

Note: Client and server applications that need to exchange a *series* of messages to complete a transaction must not use MQOO_BIND_NOT_FIXED (or MQOO_BIND_AS_Q_DEF when *DefBind* has the value MQBND_BIND_NOT_FIXED), because successive messages in the series might be sent to different instances of the server application.

If MQOO_BROWSE or one of the MQOO_INPUT_* options is specified for a cluster queue, the queue manager is forced to select the local instance of the cluster queue. As a result, the binding of the queue handle is fixed, even if MQOO_BIND_NOT_FIXED is specified.

If MQOO_INQUIRE is specified with MQOO_BIND_NOT_FIXED, successive MQINQ calls using that handle might inquire different instances of the cluster queue, although usually all the instances have the same attribute values.

MQOO_BIND_NOT_FIXED is valid only for queues, and affects only cluster queues. If specified for a queue that is not a cluster queue, the option is ignored.

MQOO_BIND_AS_Q_DEF

The local queue manager binds the queue handle in the way defined by the *DefBind* queue attribute. The value of this attribute is either MQBND_BIND_ON_OPEN or MQBND_BIND_NOT_FIXED.

MQOO_BIND_AS_Q_DEF is the default if neither MQOO_BIND_ON_OPEN nor MQOO_BIND_NOT_FIXED is specified.

MQOO_BIND_AS_Q_DEF aids program documentation. It is not intended that this option be used with either of the other two bind options, but because its value is zero such use cannot be detected.

Context options: The following options control the processing of message context:

MQOO_SAVE_ALL_CONTEXT

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 about message context, see the *WebSphere MQ Application Programming Guide*.

This context information can be passed to a message that is subsequently put on a queue using the MQPUT or MQPUT1 calls. See the MQPMO_PASS_IDENTITY_CONTEXT and MQPMO_PASS_ALL_CONTEXT options described in “MQPMO – Put-message options” on page 272.

Until a message has been successfully retrieved, context cannot be passed to a message being put on a queue.

A message retrieved using one of the MQGMO_BROWSE_* browse options does **not** have its context information saved (although the context fields in the *MsgDesc* parameter are set after a browse).

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues. One of the MQOO_INPUT_* options must be specified.

MQOO_PASS_IDENTITY_CONTEXT

This allows the MQPMO_PASS_IDENTITY_CONTEXT option to be specified in the *PutMsgOpts* parameter when a message is put on a queue; this gives the message the identity context information from an input queue that was opened with the MQOO_SAVE_ALL_CONTEXT option. For more information about message context, see the *WebSphere MQ Application Programming Guide*.

The MQOO_OUTPUT option must be specified.

This option is valid for all types of queue, including distribution lists.

MQOO_PASS_ALL_CONTEXT

This allows the MQPMO_PASS_ALL_CONTEXT option to be specified in the *PutMsgOpts* parameter when a message is put on a queue; this gives the message the identity and origin context information from an input queue that was opened with the MQOO_SAVE_ALL_CONTEXT option. For more information about message context, see the *WebSphere MQ Application Programming Guide*.

This option implies MQOO_PASS_IDENTITY_CONTEXT, which need not therefore be specified. The MQOO_OUTPUT option must be specified.

This option is valid for all types of queue, including distribution lists.

MQOO_SET_IDENTITY_CONTEXT

This allows the MQPMO_SET_IDENTITY_CONTEXT option to be specified in the *PutMsgOpts* parameter when a message is put on a queue; this gives the message the identity context information contained in the *MsgDesc* parameter specified on the MQPUT or MQPUT1 call. For more information about message context, see the *WebSphere MQ Application Programming Guide*.

This option implies MQOO_PASS_IDENTITY_CONTEXT, which need not therefore be specified. The MQOO_OUTPUT option must be specified.

This option is valid for all types of queue, including distribution lists.

MQOO_SET_ALL_CONTEXT

This allows the MQPMO_SET_ALL_CONTEXT option to be specified in the *PutMsgOpts* parameter when a message is put on a queue; this gives the message the identity and origin context information contained in the *MsgDesc* parameter specified on the MQPUT or MQPUT1 call. For more information about message context, see the *WebSphere MQ Application Programming Guide*.

This option implies the following options, which need not therefore be specified:

- MQOO_PASS_IDENTITY_CONTEXT
- MQOO_PASS_ALL_CONTEXT
- MQOO_SET_IDENTITY_CONTEXT

The MQOO_OUTPUT option must be specified.

This option is valid for all types of queue, including distribution lists.

Other options: The following options control authorization checking, what happens when the queue manager is quiescing, and whether to resolve the local queue name:

MQOO_ALTERNATE_USER_AUTHORITY

The *AlternateUserId* field in the *ObjDesc* parameter contains a user identifier to use to validate this MQOPEN call. The call can succeed only if this *AlternateUserId* is authorized to open the object with the specified access options, regardless of whether the user identifier under which the application is running is authorized to do so. This does not apply to any context options specified, however, which are always checked against the user identifier under which the application is running.

This option is valid for all types of object.

MQOO_FAIL_IF QUIESCING

The MQOPEN call fails if the queue manager is in quiescing state.

On z/OS, for a CICS or IMS application, this option also forces the MQOPEN call to fail if the connection is in quiescing state.

This option is valid for all types of object.

For information about client channels see the *WebSphere MQ Clients* book.

MQOO_RESOLVE_LOCAL_Q

Fill the *ResolvedQName* in the MQOD structure with the name of the local queue that was opened. Similarly, the *ResolvedQMgrName* is filled with the name of the local queue manager hosting the local queue. If the MQOD structure is less than Version 3, MQOO_RESOLVE_LOCAL_Q is ignored with no error being returned.

The local queue is always returned when either a local, alias, or model queue is opened, but this is not the case when, for example, a remote queue or a non-local cluster queue is opened without the MQOO_RESOLVE_LOCAL_Q option; the *ResolvedQName* and *ResolvedQMgrName* are filled with the *RemoteQName* and *RemoteQMgrName* found in the remote queue definition, or similarly with the chosen remote cluster queue.

If you specify MQOO_RESOLVE_LOCAL_Q when opening, for example, a remote queue, *ResolvedQName* is the transmission queue to which messages will be put. The *ResolvedQMgrName* is filled with the name of the local queue manager hosting the transmission queue.

If you are authorized for browse, input, or output on a queue, you have the required authority to specify this flag on the MQOPEN call. No special authority is needed.

This option is valid only for queues and queue managers.

Read ahead options: The following options control whether non-persistent messages are sent to the client before an application requests them. The following notes apply to the read ahead options:

- Only one of these options can be specified.
- These options are valid only for topics and local, alias and model queues. They are not valid for remote queues, distribution lists or queue managers.

- These options are only applicable when one of MQOO_BROWSE, MQOO_INPUT_SHARED and MQOO_INPUT_EXCLUSIVE are also specified although it is not an error to specify these options with MQOO_INQUIRE or MQOO_SET.
- If the application is not running as a WebSphere MQ client, these options are ignored.

MQOO_NO_READ_AHEAD

Non-persistent messages are not sent the client before an application requests them.

MQOO_READ_AHEAD

Non-persistent messages are sent to the client before an application requests them.

MQOO_READ_AHEAD_AS_Q_DEF

Read ahead behavior is determined by the default read ahead attribute of the queue being opened. This is the default value.

Hobj (MQHOBJ) – output

This handle represents the access that has been established to the object. It must be specified on subsequent MQ calls that operate on the object. It ceases to be valid when the MQCLOSE call is issued, or when the unit of processing that defines the scope of the handle terminates.

The scope of the object handle returned is the same as that of the connection handle specified on the call. See MQCONN - Hconn parameter for information about handle scope.

CompCode (MQLONG) – output

The completion code; it is one of the following:

MQCC_OK

Successful completion.

MQCC_WARNING

Warning (partial completion).

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_MULTIPLE_REASONS

(2136, X'858') Multiple reason codes returned.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE

(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ALIAS_BASE_Q_TYPE_ERROR
(2001, X'7D1') Alias base queue not a valid type.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_NOT_AVAILABLE
(2345, X'929') Coupling facility not available.

MQRC_CF_STRUC_AUTH_FAILED
(2348, X'92C') Coupling-facility structure authorization check failed.

MQRC_CF_STRUC_ERROR
(2349, X'92D') Coupling-facility structure not valid.

MQRC_CF_STRUC_FAILED
(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CF_STRUC_LIST_HDR_IN_USE
(2347, X'92B') Coupling-facility structure list-header in use.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CLUSTER_EXIT_ERROR
(2266, X'8DA') Cluster workload exit failed.

MQRC_CLUSTER_PUT_INHIBITED
(2268, X'8DC') Put calls inhibited for all queues in cluster.

MQRC_CLUSTER_RESOLUTION_ERROR
(2189, X'88D') Cluster name resolution failed.

MQRC_CLUSTER_RESOURCE_ERROR
(2269, X'8DD') Cluster resource error.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_DB2_NOT_AVAILABLE
(2342, X'926') DB2 subsystem not available.

MQRC_DEF_XMIT_Q_TYPE_ERROR
(2198, X'896') Default transmission queue not local.

MQRC_DEF_XMIT_Q_USAGE_ERROR
(2199, X'897') Default transmission queue usage error.

MQRC_DYNAMIC_Q_NAME_ERROR
(2011, X'7DB') Name of dynamic queue not valid.

MQRC_HANDLE_NOT_AVAILABLE
(2017, X'7E1') No more handles available.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_MULTIPLE_REASONS
(2136, X'858') Multiple reason codes returned.

MQRC_NAME_IN_USE
(2201, X'899') Name in use.

MQRC_NAME_NOT_VALID_FOR_TYPE
(2194, X'892') Object name not valid for object type.

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_OBJECT_ALREADY_EXISTS
(2100, X'834') Object already exists.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OBJECT_IN_USE
(2042, X'7FA') Object already open with conflicting options.

MQRC_OBJECT_LEVEL_INCOMPATIBLE
(2360, X'938') Object level not compatible.

MQRC_OBJECT_NAME_ERROR
(2152, X'868') Object name not valid.

MQRC_OBJECT_NOT_UNIQUE
(2343, X'927') Object not unique.

MQRC_OBJECT_Q_MGR_NAME_ERROR
(2153, X'869') Object queue-manager name not valid.

MQRC_OBJECT_RECORDS_ERROR
(2155, X'86B') Object records not valid.

MQRC_OBJECT_STRING_ERROR
(2441, X'0989') Objectstring field not valid

MQRC_OBJECT_TYPE_ERROR
(2043, X'7FB') Object type not valid.

MQRC_OD_ERROR
(2044, X'7FC') Object descriptor structure not valid.

MQRC_OPTION_NOT_VALID_FOR_TYPE
(2045, X'7FD') Option not valid for object type.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_PAGESET_FULL
(2192, X'890') External storage medium is full.

MQRC_Q_DELETED
(2052, X'804') Queue has been deleted.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR QUIESCING
(2161, X'871') Queue manager quiescing.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_Q_TYPE_ERROR
(2057, X'809') Queue type not valid.

MQRC_RECS_PRESENT_ERROR
(2154, X'86A') Number of records present not valid.

MQRC_REMOTE_Q_NAME_ERROR
(2184, X'888') Remote queue name not valid.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_RESPONSE_RECORDS_ERROR
(2156, X'86C') Response records not valid.

MQRC_SECURITY_ERROR
(2063, X'80F') Security error occurred.

MQRC_STOPPED_BY_CLUSTER_EXIT
(2188, X'88C') Call rejected by cluster workload exit.

MQRC_STORAGE_MEDIUM_FULL
(2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

MQRC_UNKNOWN_ALIAS_BASE_Q
(2082, X'822') Unknown alias base queue.

MQRC_UNKNOWN_DEF_XMIT_Q
(2197, X'895') Unknown default transmission queue.

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRC_UNKNOWN_OBJECT_Q_MGR
(2086, X'826') Unknown object queue manager.

MQRC_UNKNOWN_REMOTE_Q_MGR
(2087, X'827') Unknown remote queue manager.

MQRC_UNKNOWN_XMIT_Q
(2196, X'894') Unknown transmission queue.

MQRC_WRONG_CF_LEVEL
(2366, X'93E') Coupling-facility structure is wrong level.

MQRC_XMIT_Q_TYPE_ERROR
(2091, X'82B') Transmission queue not local.

MQRC_XMIT_Q_USAGE_ERROR
(2092, X'82C') Transmission queue with wrong usage.

For detailed information on these codes, see:

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

Usage notes for MQOPEN

General notes

1. The object opened is one of the following:

- A queue to:
 - Get or browse messages (using the MQGET call)
 - Put messages (using the MQPUT call)
 - Inquire about the attributes of the queue (using the MQINQ call)
 - Set the attributes of the queue (using the MQSET call)

If the queue named is a model queue, a dynamic local queue is created. See the *ObjDesc* parameter described in “MQOPEN – Open object” on page 505.

A distribution list is a special type of queue object that contains a list of queues. It can be opened to put messages, but not to get or browse messages, or to inquire or set attributes. See usage note 8 for further details.

A queue that has QSGDISP(GROUP) is a special type of queue definition that cannot be used with the MQOPEN or MQPUT1 calls.

- A namelist to inquire about the names of the queues in the list (using the MQINQ call).
- A process definition to inquire about the process attributes (using the MQINQ call).
- The queue manager to inquire about the attributes of the local queue manager (using the MQINQ call).
- A topic to publish a message (using the MQPUT call)

2. An application can open the same object more than once. A different object handle is returned for each open. Each handle that is returned can be used for the functions for which the corresponding open was performed.

3. If the object being opened is a queue other than a cluster queue, all name resolution within the local queue manager takes place at the time of the MQOPEN call. This can include:

- Resolution of the name of a local definition of a remote queue to the name of the remote queue manager, and the name by which the queue is known at the remote queue manager

- Resolution of the remote queue-manager name to the name of a local transmission queue
- (z/OS only) Resolution of the remote queue-manager name to the name of the shared transmission queue used by the IGQ agent (applies only if the local and remote queue managers belong to the same queue-sharing group)
- Alias resolution to the name of a base queue or a topic object.

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 or a topic object to which the alias resolves. If the object being opened is a cluster queue, name resolution can occur at the time of the MQOPEN call, or be deferred until later. The point at which resolution occurs is controlled by the MQOO_BIND_* options specified on the MQOPEN call:

- MQOO_BIND_ON_OPEN
- MQOO_BIND_NOT_FIXED
- MQOO_BIND_AS_Q_DEF

Refer to *WebSphere MQ Queue Manager Clusters* for more information about name resolution for cluster queues.

4. An MQOPEN call with the MQOO_BROWSE option establishes a browse cursor, for use with MQGET calls that specify the object handle and one of the browse options. This allows the queue to be scanned without altering its contents. A message that has been found by browsing can subsequently be removed from the queue by using the MQGMO_MSG_UNDER_CURSOR option.

Multiple browse cursors can be active for a single application by issuing several MQOPEN requests for the same queue.

5. Applications started by a trigger monitor are passed the name of the queue that is associated with the application when the application is started. This queue name can be specified in the *ObjDesc* parameter to open the queue. See “MQTMC2 – Trigger message 2 (character format)” on page 373 for further details.
6. On i5/OS, applications running in compatibility mode are connected automatically to the queue manager by the first MQOPEN call issued by the application (if the application has not already connected to the queue manager by using the MQCONN call).

Applications not running in compatibility mode must issue the MQCONN or MQCONNX call to connect to the queue manager explicitly, before using the MQOPEN call to open an object.

Read ahead options

The following notes apply to the use of read ahead options.

1. The read ahead options are applicable only when one, and only one, of the MQOO_BROWSE, MQOO_INPUT_SHARED and MQOO_INPUT_EXCLUSIVE options are also specified. An error will not be thrown if a read ahead options are specified with the MQOO_INQUIRE or MQOO_SET options.
2. Read ahead will not be enabled when requested if the options used on the first MQGET call are not supported for use with read ahead. Also, read ahead is disabled when the client is connecting to a queue manager that does not support read ahead.

3. If the application is not running as a WebSphere MQ client, read ahead options are ignored.

Cluster queues

The following notes apply to the use of cluster queues.

1. When a cluster queue is opened for the first time, and the local queue manager is not a full repository queue manager, the local queue manager obtains information about the cluster queue from a full repository queue manager. When the network is busy, it can take several seconds for the local queue manager to receive the needed information from the repository queue manager. As a result, the application issuing the MQOPEN call might have to wait for up to 10 seconds before control returns from the MQOPEN call. If the local queue manager does not receive the needed information about the cluster queue within this time, the call fails with reason code MQRC_CLUSTER_RESOLUTION_ERROR.
2. When a cluster queue is opened and there are multiple instances of the queue in the cluster, the instance opened depends on the options specified on the MQOPEN call:
 - If the options specified include any of the following:
 - MQOO_BROWSE
 - MQOO_INPUT_AS_Q_DEF
 - MQOO_INPUT_EXCLUSIVE
 - MQOO_INPUT_SHARED
 - MQOO_SETthe instance of the cluster queue opened must be the local instance. If there is no local instance of the queue, the MQOPEN call fails.
 - If the options specified include none of the above, but include one or both of the following:
 - MQOO_INQUIRE
 - MQOO_OUTPUTthe instance opened is the local instance if there is one, and a remote instance otherwise (if using the CLWLUSEQ defaults). The instance chosen by the queue manager can, however, be altered by a cluster workload exit (if there is one).
3. If there is a subscription for the queue, but it is not acknowledged by a full repository, the object is not present in the cluster and the call fails with reason code MQRC_OBJECT_NAME.

For more information about cluster queues, refer to *WebSphere MQ Queue Manager Clusters*.

Distribution lists

The following notes apply to the use of distribution lists.

Distribution lists are supported in the following environments: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

1. Fields in the MQOD structure must be set as follows when opening a distribution list:
 - *Version* must be MQOD_VERSION_2 or greater.

- *ObjectType* must be MQOT_Q.
- *ObjectName* must be blank or the null string.
- *ObjectQMgrName* must be blank or the null string.
- *RecsPresent* must be greater than zero.
- One of *ObjectRecOffset* and *ObjectRecPtr* must be zero and the other nonzero.
- No more than one of *ResponseRecOffset* and *ResponseRecPtr* can be nonzero.
- There must be *RecsPresent* object records, addressed by either *ObjectRecOffset* or *ObjectRecPtr*. The object records must be set to the names of the destination queues to be opened.
- If one of *ResponseRecOffset* and *ResponseRecPtr* is nonzero, there must be *RecsPresent* response records present. They are set by the queue manager if the call completes with reason code MQRC_MULTIPLE_REASONS.

A version-2 MQOD can also be used to open a single queue that is not in a distribution list, by ensuring that *RecsPresent* is zero.

2. Only the following open options are valid in the *Options* parameter:
 - MQOO_OUTPUT
 - MQOO_PASS_*_CONTEXT
 - MQOO_SET_*_CONTEXT
 - MQOO_ALTERNATE_USER_AUTHORITY
 - MQOO_FAIL_IF QUIESCING
3. The destination queues in the distribution list can be local, alias, or remote queues, but they cannot be model queues. If a model queue is specified, that queue fails to open, with reason code MQRC_Q_TYPE_ERROR. However, this does not prevent other queues in the list being opened successfully.
4. The completion code and reason code parameters are set as follows:
 - If the open operations for the queues in the distribution list all succeed or fail in the same way, the completion code and reason code parameters are set to describe the common result. The MQRR response records (if provided by the application) are not set in this case.
 For example, if every open succeeds, the completion code and reason code are set to MQCC_OK and MQRC_NONE respectively; if every open fails because none of the queues exists, the parameters are set to MQCC_FAILED and MQRC_UNKNOWN_OBJECT_NAME.
 - If the open operations for the queues in the distribution list do not all succeed or fail in the same way:
 - The completion code parameter is set to MQCC_WARNING if at least one open succeeded, and to MQCC_FAILED if all failed.
 - The reason code parameter is set to MQRC_MULTIPLE_REASONS.
 - The response records (if provided by the application) are set to the individual completion codes and reason codes for the queues in the distribution list.
5. When a distribution list has been opened successfully, the handle *Hobj* returned by the call can be used on subsequent MQPUT calls to put messages to queues in the distribution list, and on an MQCLOSE call to relinquish access to the distribution list. The only valid close option for a distribution list is MQCO_NONE.

The MQPUT1 call can also be used to put a message to a distribution list; the MQOD structure defining the queues in the list is specified as a parameter on that call.

6. 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 resolve to the same physical queue. If the MQOPEN or MQPUT1 call for a distribution list would cause the number of handles in use by the application to exceed *MaxHandles*, the call fails with reason code MQRC_HANDLE_NOT_AVAILABLE.
7. Each destination that is opened successfully has the value of its *OpenOutputCount* attribute incremented by one. If two or more of the destinations in the distribution list resolve to the same physical queue, that queue has its *OpenOutputCount* attribute incremented by the number of destinations in the distribution list that resolve to that queue.
8. 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.
9. A distribution list can contain only one destination.

Remote queues

The following notes apply to the use of remote queues.

A remote queue can be specified in one of two ways in the *ObjDesc* parameter of this call.

- By specifying for *ObjectName* the name of a local definition of the remote queue. In this case, *ObjectQMgrName* refers to the local queue manager, and can be specified as blanks or (in the C programming language) a null string. The security validation performed by the local queue manager verifies that the user is authorized to open the local definition of the remote queue.
- By specifying for *ObjectName* the name of the remote queue as known to the remote queue manager. In this case, *ObjectQMgrName* is the name of the remote queue manager. The security validation performed by the local queue manager verifies that the user is authorized to send messages to the transmission queue resulting from the name resolution process.

In either case:

- No messages are sent by the local queue manager to the remote queue manager to check that the user is authorized to put messages on the queue.
- When a message arrives at the remote queue manager, the remote queue manager might reject it because the user originating the message is not authorized.

See the *ObjectName* and *ObjectQMgrName* fields described in “MQOD – Object descriptor” on page 249 for more information.

Objects

Security

The following notes relate to the security aspects of using MQOPEN.

The queue manager performs security checks when an MQOPEN call is issued, to verify that the user identifier under which the application is running has the appropriate level of authority before access is permitted. The authority check is made on the name of the object being opened, and not on the name, or names, resulting after a name has been resolved.

If the object being opened is an alias queue which points at a topic object, the queue manager performs a security check on the alias queue name, before performing a security check for the topic as if the topic object had been used directly.

If the object being opened is a topic object, whether by means of *ObjectName* alone or by using the *ObjectString* (with or without a basing *ObjectName*), the queue manager performs the security check by using the resultant topic string, taken from within the topic object specified in *ObjectName*, and if required concatenating it with that provided in *ObjectString*, and then finding the closest topic object at or above that point in the topic tree to perform the security check against. This may not be the same topic object that was specified in *ObjectName*.

If the object being opened is a model queue, the queue manager performs a full security check against both the name of the model queue and the name of the dynamic queue that is created. If the resulting dynamic queue is subsequently opened explicitly, a further resource security check is performed against the name of the dynamic queue.

On z/OS, the queue manager performs security checks only if security is enabled. For more information on security checking, see the *WebSphere MQ for z/OS System Setup Guide*.

Attributes

The following notes relate to attributes.

The attributes of an object can change while an application has the object open. In many cases, the application does not notice this, but for certain attributes the queue manager marks the handle as no longer valid. These are:

- Any attribute that affects the name resolution of the object. This applies regardless of the open options used, and includes the following:
 - A change to the *BaseQName* attribute of an alias queue that is open.
 - A change to the *TargetType* attribute of an alias queue that is open.
 - A change to the *RemoteQName* or *RemoteQMGrName* queue attributes, for any handle that is open for this queue, or for a queue that resolves through this definition as a queue-manager alias.
 - Any change that causes a currently-open handle for a remote queue to resolve to a different *transmission* queue, or to fail to resolve to one at all. For example, this can include:
 - A change to the *XmitQName* attribute of the local definition of a remote queue, whether the definition is being used for a queue, or for a queue-manager alias.
 - (z/OS only) A change to the value of the *IntraGroupQueuing* queue-manager attribute, or a change in the definition of the shared transmission queue (SYSTEM.QSG.TRANSMIT.QUEUE) used by the IGQ agent.

There is one exception to this: the creation of a new transmission queue. A handle that would have resolved to this queue had it been present when the handle was opened, but instead resolved to the default transmission queue, is not made invalid.

- A change to the *DefXmitQName* queue-manager attribute. In this case all open handles that resolved to the previously-named queue (that resolved to it only because it was the default transmission queue) are marked as invalid. Handles that resolved to this queue for other reasons are not affected.
- The *Shareability* queue attribute, if there are two or more handles that are currently providing MQOO_INPUT_SHARED access for this queue, or for a queue that resolves to this queue. If this is the case, *all* handles that are open for this queue, or for a queue that resolves to this queue, are marked as invalid, regardless of the open options.
On z/OS, the handles described above are marked as invalid if one or more handles is currently providing MQOO_INPUT_SHARED or MQOO_INPUT_EXCLUSIVE access to the queue.
- The *Usage* queue attribute, for all handles that are open for this queue, or for a queue that resolves to this queue, regardless of the open options.

When a handle is marked as invalid, all subsequent calls (other than MQCLOSE) using this handle fail with reason code MQRC_OBJECT_CHANGED. The application must 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 causes this to happen, use a special *force* version of the command.

Language invocations for MQOPEN

The MQOPEN call is supported in the programming languages shown below.

C invocation

```
MQOPEN (Hconn, &ObjDesc, Options, &Hobj, &CompCode,
        &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;      /* Connection handle */
MQOD    ObjDesc;    /* Object descriptor */
MQLONG  Options;    /* Options that control the action of MQOPEN */
MQHOBJ  Hobj;       /* Object handle */
MQLONG  CompCode;   /* Completion code */
MQLONG  Reason;     /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQOPEN' USING HCONN, OBJDESC, OPTIONS, HOBJ, COMPCODE,
                   REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN    PIC S9(9) BINARY.
** Object descriptor
01 OBJDESC.
   COPY CMQODV.
** Options that control the action of MQOPEN
01 OPTIONS  PIC S9(9) BINARY.
** Object handle
```

```

01 HOBJ      PIC S9(9) BINARY.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON   PIC S9(9) BINARY.

```

PL/I invocation

```
call MQOPEN (Hconn, ObjDesc, Options, Hobj, CompCode, Reason);
```

Declare the parameters as follows:

```

dcl Hconn      fixed bin(31); /* Connection handle */
dcl ObjDesc    like MQOD;    /* Object descriptor */
dcl Options    fixed bin(31); /* Options that control the action of
                               MQOPEN */
dcl Hobj       fixed bin(31); /* Object handle */
dcl CompCode   fixed bin(31); /* Completion code */
dcl Reason     fixed bin(31); /* Reason code qualifying CompCode */

```

System/390 assembler invocation

```
CALL MQOPEN,(HCONN,OBJDESC,OPTIONS,HOBJ,COMPCODE,REASON)
```

Declare the parameters as follows:

```

HCONN      DS      F Connection handle
OBJDESC    CMQODA  , Object descriptor
OPTIONS    DS      F Options that control the action of MQOPEN
HOBJ       DS      F Object handle
COMPCODE   DS      F Completion code
REASON     DS      F Reason code qualifying COMPCODE

```

Visual Basic invocation

```
MQOPEN Hconn, ObjDesc, Options, Hobj, CompCode, Reason
```

Declare the parameters as follows:

```

Dim Hconn      As Long 'Connection handle'
Dim ObjDesc    As MQOD 'Object descriptor'
Dim Options    As Long 'Options that control the action of MQOPEN'
Dim Hobj       As Long 'Object handle'
Dim CompCode   As Long 'Completion code'
Dim Reason     As Long 'Reason code qualifying CompCode'

```

MQPUT – Put message

The MQPUT call puts a message on a queue or distribution list, or to a topic. The queue, distribution list or topic must already be open.

Syntax for MQPUT

```
MQPUT (Hconn, Hobj, MsgDesc, PutMsgOpts, BufferLength,
      Buffer, CompCode, Reason)
```

Parameters for MQPUT

The MQPUT call has the following parameters.

Hconn (MQHCONN) – input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNEX call.

On z/OS for CICS applications, and on i5/OS for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN

Default connection handle.

Hobj (MQHOBJ) – input

This handle represents the queue to which the message is added, or the topic to which the message is published. The value of *Hobj* was returned by a previous MQOPEN call that specified the MQOO_OUTPUT option.

MsgDesc (MQMD) – input/output

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 181 for details.

If the application provides a version-1 MQMD, the message data can be prefixed with an MQMDE structure to specify values for the fields that exist in the version-2 MQMD but not the version-1. The *Format* field in the MQMD must be set to MQFMT_MD_EXTENSION to indicate that an MQMDE is present. See “MQMDE – Message descriptor extension” on page 239 for more details.

The application does not need to provide an MQMD structure provided that a valid message handle is supplied in the *OriginalMsgHandle* or *NewMsgHandle* fields of the MQPMO structure. If nothing is provided in one of these fields, the descriptor of the message is taken from the descriptor associated with the message handles.

PutMsgOpts (MQPMO) – input/output

See “MQPMO – Put-message options” on page 272 for details.

BufferLength (MQLONG) – input

The length of the message in *Buffer*. Zero is valid, and indicates that the message contains no application data. The upper limit for *BufferLength* depends on various factors:

- If the destination is a local queue or resolves to a local queue, the upper limit depends on whether:
 - The local queue manager supports segmentation.
 - The sending application specifies the flag that allows the queue manager to segment the message. This flag is MQMF_SEGMENTATION_ALLOWED, and can be specified either in a version-2 MQMD, or in an MQMDE used with a version-1 MQMD.

If both of these conditions are satisfied, *BufferLength* cannot exceed 999 999 999 minus the value of the *Offset* field in MQMD. The longest logical message that can be put is therefore 999 999 999 bytes (when *Offset* is zero). However, resource constraints imposed by the operating system or environment in which the application is running might result in a lower limit.

If one or both of the above conditions is not satisfied, *BufferLength* cannot exceed the smaller of the queue’s *MaxMsgLength* attribute and queue-manager’s *MaxMsgLength* attribute.

- If the destination is a remote queue or resolves to a remote queue, the conditions for local queues apply, *but at each queue manager through which the message must pass in order to reach the destination queue*; in particular:
 1. The local transmission queue used to store the message temporarily at the local queue manager
 2. Intermediate transmission queues (if any) used to store the message at queue managers on the route between the local and destination queue managers
 3. The destination queue at the destination queue manager

The longest message that can be put is therefore governed by the most restrictive of these queues and queue managers.

When a message is on a transmission queue, additional information resides with the message data, and this reduces the amount of application data that can be carried. In this situation, subtract MQ_MSG_HEADER_LENGTH bytes from the *MaxMsgLength* values of the transmission queues when determining the limit for *BufferLength*.

Note: Only failure to comply with condition 1 can be diagnosed synchronously (with reason code MQRC_MSG_TOO_BIG_FOR_Q or MQRC_MSG_TOO_BIG_FOR_Q_MGR) when the message is put. If conditions 2 or 3 are not satisfied, the message is redirected to a dead-letter (undelivered-message) queue, either at an intermediate queue manager or at the destination queue manager. If this happens, a report message is generated if one was requested by the sender.

Buffer (MQBYTEExBufferLength) – input

This is a buffer containing the application data to be sent. The buffer must be aligned on a boundary appropriate to the nature of the data in the message. 4-byte alignment is suitable for most messages (including messages containing MQ header structures), but some messages might require more stringent alignment. For example, a message containing a 64-bit binary integer might require 8-byte alignment.

If *Buffer* contains character or numeric data, set the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter to the values appropriate to the data; this enables the receiver of the message to convert the data (if necessary) to the character set and encoding used by the receiver.

Note: All the other parameters on the MQPUT call must be in the character set and encoding of the local queue manager (given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE).

In the C programming language, the parameter is declared as a pointer-to-void; the address of any type of data can be specified as the parameter.

If the *BufferLength* parameter is zero, *Buffer* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler can be null.

CompCode (MQLONG) – output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

| MQCC_WARNING
| Warning (partial completion).

| MQCC_FAILED
| Call failed.

| Reason (MQLONG) – output

| The reason code qualifying *CompCode*.

| If *CompCode* is MQCC_OK:

| MQRC_NONE
| (0, X'000') No reason to report.

| If *CompCode* is MQCC_WARNING:

| MQRC_INCOMPLETE_GROUP
| (2241, X'8C1') Message group not complete.

| MQRC_INCOMPLETE_MSG
| (2242, X'8C2') Logical message not complete.

| MQRC_INCONSISTENT_PERSISTENCE
| (2185, X'889') Inconsistent persistence specification.

| MQRC_INCONSISTENT_UOW
| (2245, X'8C5') Inconsistent unit-of-work specification.

| MQRC_MULTIPLE_REASONS
| (2136, X'858') Multiple reason codes returned.

| MQRC_PRIORITY_EXCEEDS_MAXIMUM
| (2049, X'801') Message Priority exceeds maximum value supported.

| MQRC_UNKNOWN_REPORT_OPTION
| (2104, X'838') Report option(s) in message descriptor not recognized.

| If *CompCode* is MQCC_FAILED:

| MQRC_ADAPTER_NOT_AVAILABLE
| (2204, X'89C') Adapter not available.

| MQRC_ADAPTER_SERV_LOAD_ERROR
| (2130, X'852') Unable to load adapter service module.

| MQRC_ALIAS_TARGTYPE_CHANGED
| (2480, X'09B0') Subscription target type has changed from queue to topic
| object.

| MQRC_API_EXIT_ERROR
| (2374, X'946') API exit failed.

| MQRC_API_EXIT_LOAD_ERROR
| (2183, X'887') Unable to load API exit.

| MQRC_ASID_MISMATCH
| (2157, X'86D') Primary and home ASIDs differ.

| MQRC_BACKED_OUT
| (2003, X'7D3') Unit of work backed out.

| MQRC_BUFFER_ERROR
| (2004, X'7D4') Buffer parameter not valid.

| **MQRC_BUFFER_LENGTH_ERROR**
| (2005, X'7D5') Buffer length parameter not valid.

| **MQRC_CALL_IN_PROGRESS**
| (2219, X'8AB') MQI call entered before previous call complete.

| **MQRC_CF_STRUC_FAILED**
| (2373, X'945') Coupling-facility structure failed.

| **MQRC_CF_STRUC_IN_USE**
| (2346, X'92A') Coupling-facility structure in use.

| **MQRC_CFGR_ERROR**
| (2416, X'970') PCF group parameter structure MQCFGR in the message
| data is not valid.

| **MQRC_CFH_ERROR**
| (2235, X'8BB') PCF header structure not valid.

| **MQRC_CFIF_ERROR**
| (2414, X'96E') PCF integer filter parameter structure in the message data is
| not valid.

| **MQRC_CFIL_ERROR**
| (2236, X'8BC') PCF integer list parameter structure or PCIF*64 integer list
| parameter structure not valid.

| **MQRC_CFIN_ERROR**
| (2237, X'8BD') PCF integer parameter structure or PCIF*64 integer
| parameter structure not valid.

| **MQRC_CFSF_ERROR**
| (2415, X'96F') PCF string filter parameter structure in the message data is
| not valid.

| **MQRC_CFSL_ERROR**
| (2238, X'8BE') PCF string list parameter structure not valid.

| **MQRC_CFST_ERROR**
| (2239, X'8BF') PCF string parameter structure not valid.

| **MQRC_CICS_WAIT_FAILED**
| (2140, X'85C') Wait request rejected by CICS.

| **MQRC_CLUSTER_EXIT_ERROR**
| (2266, X'8DA') Cluster workload exit failed.

| **MQRC_CLUSTER_RESOLUTION_ERROR**
| (2189, X'88D') Cluster name resolution failed.

| **MQRC_CLUSTER_RESOURCE_ERROR**
| (2269, X'8DD') Cluster resource error.

| **MQRC_COD_NOT_VALID_FOR_XCF_Q**
| (2106, X'83A') COD report option not valid for XCF queue.

| **MQRC_CONNECTION_BROKEN**
| (2009, X'7D9') Connection to queue manager lost.

| **MQRC_CONNECTION_NOT_AUTHORIZED**
| (2217, X'8A9') Not authorized for connection.

| **MQRC_CONNECTION_QUIESCING**
| (2202, X'89A') Connection quiescing.

| **MQRC_CONNECTION_STOPPING**
| (2203, X'89B') Connection shutting down.

| **MQRC_CONTEXT_HANDLE_ERROR**
| (2097, X'831') Queue handle referred to does not save context.

| **MQRC_CONTEXT_NOT_AVAILABLE**
| (2098, X'832') Context not available for queue handle referred to.

| **MQRC_DATA_LENGTH_ERROR**
| (2010, X'7DA') Data length parameter not valid.

| **MQRC_DH_ERROR**
| (2135, X'857') Distribution header structure not valid.

| **MQRC_DLH_ERROR**
| (2141, X'85D') Dead letter header structure not valid.

| **MQRC_EPH_ERROR**
| (2420, X'974') Embedded PCF structure not valid.

| **MQRC_EXPIRY_ERROR**
| (2013, X'7DD') Expiry time not valid.

| **MQRC_FEEDBACK_ERROR**
| (2014, X'7DE') Feedback code not valid.

| **MQRC_GLOBAL_UOW_CONFLICT**
| (2351, X'92F') Global units of work conflict.

| **MQRC_GROUP_ID_ERROR**
| (2258, X'8D2') Group identifier not valid.

| **MQRC_HANDLE_IN_USE_FOR_UOW**
| (2353, X'931') Handle in use for global unit of work.

| **MQRC_HCONN_ERROR**
| (2018, X'7E2') Connection handle not valid.

| **MQRC_HEADER_ERROR**
| (2142, X'85E') MQ header structure not valid.

| **MQRC_HOBJ_ERROR**
| (2019, X'7E3') Object handle not valid.

| **MQRC_IH_ERROR**
| (2148, X'864') IMS information header structure not valid.

| **MQRC_INCOMPLETE_GROUP**
| (2241, X'8C1') Message group not complete.

| **MQRC_INCOMPLETE_MSG**
| (2242, X'8C2') Logical message not complete.

| **MQRC_INCONSISTENT_PERSISTENCE**
| (2185, X'889') Inconsistent persistence specification.

| **MQRC_INCONSISTENT_UOW**
| (2245, X'8C5') Inconsistent unit-of-work specification.

| **MQRC_LOCAL_UOW_CONFLICT**
| (2352, X'930') Global unit of work conflicts with local unit of work.

| **MQRC_MD_ERROR**
| (2026, X'7EA') Message descriptor not valid.

| **MQRC_MDE_ERROR**
 | (2248, X'8C8') Message descriptor extension not valid.

| **MQRC_MISSING_REPLY_TO_Q**
 | (2027, X'7EB') Missing reply-to queue or MQPMO_SUPPRESS_REPLYTO
 | was used

| **MQRC_MISSING_WIH**
 | (2332, X'91C') Message data does not begin with MQWIH.

| **MQRC_MSG_FLAGS_ERROR**
 | (2249, X'8C9') Message flags not valid.

| **MQRC_MSG_SEQ_NUMBER_ERROR**
 | (2250, X'8CA') Message sequence number not valid.

| **MQRC_MSG_TOO_BIG_FOR_Q**
 | (2030, X'7EE') Message length greater than maximum for queue.

| **MQRC_MSG_TOO_BIG_FOR_Q_MGR**
 | (2031, X'7EF') Message length greater than maximum for queue manager.

| **MQRC_MSG_TYPE_ERROR**
 | (2029, X'7ED') Message type in message descriptor not valid.

| **MQRC_MULTIPLE_REASONS**
 | (2136, X'858') Multiple reason codes returned.

| **MQRC_NO_DESTINATIONS_AVAILABLE**
 | (2270, X'8DE') No destination queues available.

| **MQRC_NOT_OPEN_FOR_OUTPUT**
 | (2039, X'7F7') Queue not open for output.

| **MQRC_NOT_OPEN_FOR_PASS_ALL**
 | (2093, X'82D') Queue not open for pass all context.

| **MQRC_NOT_OPEN_FOR_PASS_IDENT**
 | (2094, X'82E') Queue not open for pass identity context.

| **MQRC_NOT_OPEN_FOR_SET_ALL**
 | (2095, X'82F') Queue not open for set all context.

| **MQRC_NOT_OPEN_FOR_SET_IDENT**
 | (2096, X'830') Queue not open for set identity context.

| **MQRC_OBJECT_CHANGED**
 | (2041, X'7F9') Object definition changed since opened.

| **MQRC_OBJECT_DAMAGED**
 | (2101, X'835') Object damaged.

| **MQRC_OFFSET_ERROR**
 | (2251, X'8CB') Message segment offset not valid.

| **MQRC_OPEN_FAILED**
 | (2137, X'859') Object not opened successfully.

| **MQRC_OPTIONS_ERROR**
 | (2046, X'7FE') Options not valid or not consistent.

| **MQRC_ORIGINAL_LENGTH_ERROR**
 | (2252, X'8CC') Original length not valid.

| **MQRC_PAGESET_ERROR**
 | (2193, X'891') Error accessing page-set data set.

| **MQRC_PAGESET_FULL**
| (2192, X'890') External storage medium is full.

| **MQRC_PCF_ERROR**
| (2149, X'865') PCF structures not valid.

| **MQRC_PERSISTENCE_ERROR**
| (2047, X'7FF') Persistence not valid.

| **MQRC_PERSISTENT_NOT_ALLOWED**
| (2048, X'800') Queue does not support persistent messages.

| **MQRC_PMO_ERROR**
| (2173, X'87D') Put-message options structure not valid.

| **MQRC_PMO_RECORD_FLAGS_ERROR**
| (2158, X'86E') Put message record flags not valid.

| **MQRC_PRIORITY_ERROR**
| (2050, X'802') Message priority not valid.

| **MQRC_PUT_INHIBITED**
| (2051, X'803') Put calls inhibited for the queue, for the queue to which this
| queue resolves, or the topic.

| **MQRC_PUT_MSG_RECORDS_ERROR**
| (2159, X'86F') Put message records not valid.

| **MQRC_PUT_NOT_RETAINED**
| (2479, X'09AF') Publication could not be retained

| **MQRC_Q_DELETED**
| (2052, X'804') Queue has been deleted.

| **MQRC_Q_FULL**
| (2053, X'805') Queue already contains maximum number of messages.

| **MQRC_Q_MGR_NAME_ERROR**
| (2058, X'80A') Queue manager name not valid or not known.

| **MQRC_Q_MGR_NOT_AVAILABLE**
| (2059, X'80B') Queue manager not available for connection.

| **MQRC_Q_MGR QUIESCING**
| (2161, X'871') Queue manager quiescing.

| **MQRC_Q_MGR STOPPING**
| (2162, X'872') Queue manager shutting down.

| **MQRC_Q_SPACE_NOT_AVAILABLE**
| (2056, X'808') No space available on disk for queue.

| **MQRC_RECS_PRESENT_ERROR**
| (2154, X'86A') Number of records present not valid.

| **MQRC_REPORT_OPTIONS_ERROR**
| (2061, X'80D') Report options in message descriptor not valid.

| **MQRC_RESOURCE_PROBLEM**
| (2102, X'836') Insufficient system resources available.

| **MQRC_RESPONSE_RECORDS_ERROR**
| (2156, X'86C') Response records not valid.

| **MQRC_RFH_ERROR**
| (2334, X'91E') MQRFH or MQRFH2 structure not valid.

MQRC_RMH_ERROR
 (2220, X'8AC') Reference message header structure not valid.

MQRC_SEGMENT_LENGTH_ZERO
 (2253, X'8CD') Length of data in message segment is zero.

MQRC_SEGMENTS_NOT_SUPPORTED
 (2365, X'93D') Segments not supported.

MQRC_STOPPED_BY_CLUSTER_EXIT
 (2188, X'88C') Call rejected by cluster workload exit.

MQRC_STORAGE_CLASS_ERROR
 (2105, X'839') Storage class error.

MQRC_STORAGE_MEDIUM_FULL
 (2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE
 (2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
 (2109, X'83D') Call suppressed by exit program.

MQRC_SYNCPOINT_LIMIT_REACHED
 (2024, X'7E8') No more messages can be handled within current unit of work.

MQRC_SYNCPOINT_NOT_AVAILABLE
 (2072, X'818') Syncpoint support not available.

MQRC_TM_ERROR
 (2265, X'8D9') Trigger message structure not valid.

MQRC_TMC_ERROR
 (2191, X'88F') Character trigger message structure not valid.

MQRC_UNEXPECTED_ERROR
 (2195, X'893') Unexpected error occurred.

MQRC_UOW_ENLISTMENT_ERROR
 (2354, X'932') Enlistment in global unit of work failed.

MQRC_UOW_MIX_NOT_SUPPORTED
 (2355, X'933') Mixture of unit-of-work calls not supported.

MQRC_UOW_NOT_AVAILABLE
 (2255, X'8CF') Unit of work not available for the queue manager to use.

MQRC_WIH_ERROR
 (2333, X'91D') MQWIH structure not valid.

MQRC_WRONG_MD_VERSION
 (2257, X'8D1') Wrong version of MQMD supplied.

MQRC_XQH_ERROR
 (2260, X'8D4') Transmission queue header structure not valid.

For detailed information on these codes, see:

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

Usage notes for MQPUT

Topics

The following notes apply to the use of topics:

1. When using MQPUT to publish messages on a topic, where one or more subscribers to that topic cannot be given the publication due to a problem with their subscriber queue (for example it is full), the Reason code returned to the MQPUT call and the delivery behaviour is dependant on the setting of the PMSGDLV or NPMSGDLV attributes on the TOPIC. Note that delivery of a publication to the dead letter queue when MQRO_DEAD_LETTER_Q is specified, or discarding the message when MQRO_DISCARD_MSG is specified, is considered as a successful delivery of the message. If none of the publications were delivered, the MQPUT will return with MQRC_PUBLICATION_FAILURE. This can happen in the following cases:

- A message is published to a TOPIC with PMSGDLV or NPMSGDLV (depending on the persistence of the message) set to ALL and any subscription (durable or not) has a queue which cannot receive the publication.
- A message is published to a TOPIC with PMSGDLV or NPMSGDLV (depending on the persistence of the message) set to ALLDUR and a durable subscription has a queue which cannot receive the publication.

The MQPUT can return with MQRC_NONE even though publications could not be delivered to some subscribers in the following cases:

- A message is published to a TOPIC with PMSGDLV or NPMSGDLV (depending on the persistence of the message) set to ALLAVAIL and any subscription, durable or not, has a queue which cannot receive the publication.
 - A message is published to a TOPIC with PMSGDLV or NPMSGDLV (depending on the persistence of the message) set to ALLDUR and a non-durable subscription has a queue which cannot receive the publication.
2. If there are no subscribers to the topic being used, the message published is not sent to any queue and is discarded. It does not matter whether the message is persistent or non-persistent, or whether it has unlimited expiry or has an expiry time, it is still discarded if there are no subscribers. The exception to this is if the message is to be retained, in which case, although it is not sent to any subscribers' queues, it is stored against the topic to be delivered to any new subscriptions or to any subscribers that ask for retained publications using MQSUBRQ.

MQPUT and MQPUT1

You can use both the MQPUT and MQPUT1 calls to put messages on a queue; which call to use depends on the circumstances

- Use the MQPUT call to place multiple messages on the *same* queue.
An MQOPEN call specifying the MQOO_OUTPUT option is issued first, followed by one or more MQPUT requests to add messages to the queue; finally the queue is closed with an MQCLOSE call. This gives better performance than repeated use of the MQPUT1 call.
- Use the MQPUT1 call to put only *one* message 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.

Destination Queues

The following notes apply to the use of destination queues:

1. If an application puts a sequence of messages on the same queue without using message groups, the order of those messages is preserved provided that the conditions detailed below are satisfied. Some conditions apply to both local and remote destination queues; other conditions apply only to remote destination queues.

Conditions that apply to local and remote destination queues

- All the MQPUT calls are within the same unit of work, or none of them is within a unit of work.

Be aware that when messages are put onto a particular queue within a single unit of work, messages from other applications might be interspersed with the sequence of messages on the queue.

- All 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 that the calls are made from the same application. The meaning of *same application* is determined by the environment:

- On z/OS, the application is:
 - For CICS, the CICS task
 - For IMS, the task
 - For z/OS batch, the task
- On i5/OS, the application is the job.
- On Windows and UNIX systems, the application is the thread.

- The messages all have the same priority.
- The messages are not put to a cluster queue with MQOO_BIND_NOT_FIXED specified (or with MQOO_BIND_AS_Q_DEF in effect when the DefBind queue attribute has the value MQBND_BIND_NOT_FIXED).

Additional conditions that apply to remote destination queues

- There is only one path from the sending queue manager to the destination queue manager.

If some messages in the sequence might go on a different path (for example, because of reconfiguration, traffic balancing, or path selection based on message size), the order of the messages at the destination queue manager cannot be guaranteed.

- Messages are not placed temporarily on dead-letter queues at the sending, intermediate, or destination queue managers.

If one or more of the messages is put temporarily on a dead-letter queue (for example, because a transmission queue or the destination queue is temporarily full), the messages can arrive on the destination queue out of sequence.

- The messages are either all persistent or all nonpersistent.

If a channel on the route between the sending and destination queue managers has its *NonPersistentMsgSpeed* attribute set to MQNPMMS_FAST, nonpersistent messages can jump ahead of persistent messages, resulting in the order of persistent messages relative to nonpersistent messages not being preserved. However, the order of persistent messages relative to each other, and of nonpersistent messages relative to each other, is preserved.

If these conditions are not satisfied, you can use message groups to preserve message order, but this requires both the sending and receiving applications to use the message-grouping support. For more information about message groups, see:

- MQMD - MsgFlags field
- MQPMO_LOGICAL_ORDER
- MQGMO_LOGICAL_ORDER

Distribution Lists

The following notes apply to the use of distribution lists.

Distribution lists are supported in the following environments: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

1. You can put messages to a distribution list using either a version-1 or a version-2 MQPMO. If you use a version-1 MQPMO (or a version-2 MQPMO with *RecsPresent* equal to zero), the application can provide no put message records or response records. You cannot identify the queues that encounter errors if the message is sent successfully to some queues in the distribution list and not others.

If the application provides put message records or response records, set the *Version* field to MQPMO_VERSION_2.

You can also use a version-2 MQPMO to send messages to a single queue that is not in a distribution list, by ensuring that *RecsPresent* is zero.

2. 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 MQRRC response records (if provided by the application) are not set in this case.

For example, if every put succeeds, the completion code and reason code are set to MQCC_OK and MQRC_NONE; if every put fails because all the queues are inhibited for puts, the parameters are set to MQCC_FAILED and MQRC_PUT_INHIBITED.

- If the puts to the queues in the distribution list do not all succeed or fail in the same way:
 - The completion code parameter is set to MQCC_WARNING if at least one put succeeded, and to MQCC_FAILED if all failed.
 - The reason code parameter is set to MQRC_MULTIPLE_REASONS.
 - The response records (if provided by the application) are set to the individual completion codes and reason codes for the queues in the distribution list.

If the put to a destination fails because the open for that destination failed, the fields in the response record are set to MQCC_FAILED and MQRC_OPEN_FAILED; that destination is included in *InvalidDestCount*.

3. 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 can 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 DistLists).

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 into smaller distribution-list messages, each containing fewer destinations. If the application message data only just fits on the queue, distribution-list messages cannot be used at all, and the queue manager generates one copy of the message in normal form for each destination that uses that transmission queue.

If different destinations have different message priority or message persistence (this can occur when the application specifies MQPRI_PRIORITY_AS_Q_DEF or MQPER_PERSISTENCE_AS_Q_DEF), the messages are not held in the same distribution-list message. Instead, the queue manager generates as many distribution-list messages as are necessary to accommodate the differing priority and persistence values.

4. A put to a distribution list can 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.

5. 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.

Headers

If a message is put with one or more MQ header structures at the beginning of the application message data, the queue manager performs certain checks on the header structures to verify that they are valid. If the queue manager detects an error, the call fails with an appropriate reason code. The checks performed vary according to the particular structures that are present:

- Checks are performed only if a version-2 or later MQMD is used on the MQPUT or MQPUT1 call. Checks are not performed if a version-1 MQMD is used, even if an MQMDE is present at the start of the message data.
- Structures that are not supported by the local queue manager, and structures following the first MQDLH in the message, are not validated.
- The MQDH and MQMDE structures are validated completely by the queue manager.
- Other structures are validated partially by the queue manager (not all fields are checked).

General checks performed by the queue manager include the following:

- The *StrucId* field must be valid.
- The *Version* field must be valid.
- The *StrucLength* field must specify a value that is large enough to include the structure plus any variable-length data that forms part of the structure.
- The *CodedCharSetId* field must not be zero, or a negative value that is not valid (MQCCSI_DEFAULT, MQCCSI_EMBEDDED, MQCCSI_Q_MGR, and MQCCSI_UNDEFINED are *not* valid in most MQ header structures).
- The *BufferLength* parameter of the call must specify a value that is large enough to include the structure (the structure must not extend beyond the end of the message).

In addition to general checks on structures, the following conditions must be satisfied:

- The sum of the lengths of the structures in a PCF message must equal the length specified by the *BufferLength* parameter on the MQPUT or MQPUT1 call. A PCF message is a message that has a format name of MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF.
- An MQ structure must not be truncated, except in the following situations where truncated structures are permitted:
 - Messages that are report messages.
 - PCF messages.
 - Messages containing an MQDLH structure. (Structures *following* the first MQDLH can be truncated; structures preceding the MQDLH cannot.)
- An MQ structure must not be split over two or more segments; the structure must be contained entirely within one segment.

Buffer

For the Visual Basic programming language, the following points apply:

- If the size of the *Buffer* parameter is less than the length specified by the *BufferLength* parameter, the call fails with reason code MQRC_BUFFER_LENGTH_ERROR.
- The *Buffer* parameter is declared as being of type String. If the data to be placed on the queue is not of type String, use the MQPUTAny call in place of MQPUT.

The MQPUTAny call has the same parameters as the MQPUT call, except that the *Buffer* parameter is declared as being of type Any, allowing any type of data to be placed on the queue. However, this means that *Buffer* cannot be checked to ensure that it is at least *BufferLength* bytes in size.

Language invocations for MQPUT

The MQPUT call is supported in the programming languages shown below.

C invocation

```
MQPUT (Hconn, Hobj, &MsgDesc, &PutMsgOpts, BufferLength, Buffer,  
      &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;      /* Connection handle */  
MQHOBJ  Hobj;      /* Object handle */  
MQMD    MsgDesc;   /* Message descriptor */  
MQPMO   PutMsgOpts; /* Options that control the action of MQPUT */  
MQLONG  BufferLength; /* Length of the message in Buffer */  
MQBYTE  Buffer[n];  /* Message data */  
MQLONG  CompCode;  /* Completion code */  
MQLONG  Reason;    /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQPUT' USING HCONN, HOBJ, MSGDESC, PUTMSGOPTS, BUFFERLENGTH,  
                  BUFFER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle  
01 HCONN          PIC S9(9) BINARY.  
** Object handle  
01 HOBJ           PIC S9(9) BINARY.  
** Message descriptor  
01 MSGDESC.  
   COPY CMQMDV.  
** Options that control the action of MQPUT  
01 PUTMSGOPTS.  
   COPY CMQPMOV.  
** Length of the message in BUFFER  
01 BUFFERLENGTH  PIC S9(9) BINARY.  
** Message data  
01 BUFFER        PIC X(n).  
** Completion code  
01 COMPCODE      PIC S9(9) BINARY.  
** Reason code qualifying COMPCODE  
01 REASON        PIC S9(9) BINARY.
```

PL/I invocation

```
call MQPUT (Hconn, Hobj, MsgDesc, PutMsgOpts, BufferLength, Buffer,  
           CompCode, Reason);
```

Declare the parameters as follows:

```
dc1 Hconn          fixed bin(31); /* Connection handle */  
dc1 Hobj           fixed bin(31); /* Object handle */  
dc1 MsgDesc        like MQMD;    /* Message descriptor */  
dc1 PutMsgOpts     like MQPMO;   /* Options that control the action of  
                                MQPUT */  
dc1 BufferLength    fixed bin(31); /* Length of the message in Buffer */  
dc1 Buffer          char(n);      /* Message data */  
dc1 CompCode       fixed bin(31); /* Completion code */  
dc1 Reason         fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
CALL MQPUT, (HCONN,HOBJ,MSGDESC,PUTMSGOPTS,BUFFERLENGTH,      X  
            BUFFER,COMPCODE,REASON)
```

Declare the parameters as follows:

HCONN	DS	F	Connection handle
HOBJ	DS	F	Object handle
MSGDESC	CMQMDA	,	Message descriptor
PUTMSGOPTS	CMQPMOA	,	Options that control the action of MQPUT
BUFFERLENGTH	DS	F	Length of the message in BUFFER
BUFFER	DS	CL(n)	Message data
COMPCODE	DS	F	Completion code
REASON	DS	F	Reason code qualifying COMPCODE

Visual Basic invocation

```
MQPUT Hconn, Hobj, MsgDesc, PutMsgOpts, BufferLength, Buffer, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn As Long 'Connection handle'
Dim Hobj As Long 'Object handle'
Dim MsgDesc As MQMD 'Message descriptor'
Dim PutMsgOpts As MQPMO 'Options that control the action of MQPUT'
Dim BufferLength As Long 'Length of the message in Buffer'
Dim Buffer As String 'Message data'
Dim CompCode As Long 'Completion code'
Dim Reason As Long 'Reason code qualifying CompCode'
```

MQPUT1 – Put one message

The MQPUT1 call puts one message on a queue, or distribution list, or to a topic.

The queue, distribution list, or topic does not need to be open.

Syntax for MQPUT1

MQPUT1 (*Hconn, ObjDesc, MsgDesc, PutMsgOpts, BufferLength, Buffer, CompCode, Reason*)

Parameters for MQPUT1

The MQPUT1 call has the following parameters.

Hconn (MQHCONN) – input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on i5/OS for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN

Default connection handle.

ObjDesc (MQOD) – input/output

This is a structure that identifies the queue to which the message is added, or the topic to which the message is published. See “MQOD – Object descriptor” on page 249 for details.

If the structure is a queue, the user must be authorized to open the queue for output. The queue must **not** be a model queue.

MsgDesc (MQMD) – input/output

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 181 for details.

If the application provides a version-1 MQMD, the message data can be prefixed with an MQMDE structure to specify values for the fields that exist in the version-2 MQMD but not the version-1. Set the *Format* field in the MQMD to MQFMT_MD_EXTENSION to indicate that an MQMDE is present. See “MQMDE – Message descriptor extension” on page 239 for more details.

The application does not need to provide an MQMD structure provided that a valid message handle is supplied in the *MsgHandle* field of the MQGMO structure or in the *OriginalMsgHandle* or *NewMsgHandle* fields of the MQPMO structure. If nothing is provided in one of these fields, the descriptor of the message is taken from the descriptor associated with the message handles.

PutMsgOpts (MQPMO) – input/output

See “MQPMO – Put-message options” on page 272 for details.

BufferLength (MQLONG) – input

The length of the message in *Buffer*. Zero is valid, and indicates that the message contains no application data. The upper limit depends on various factors; see the description of the *BufferLength* parameter of the MQPUT call for further details.

Buffer (MQBYTExBufferLength) – input

This is a buffer containing the application message data to be sent. Align the buffer on a boundary appropriate to the nature of the data in the message. 4-byte alignment is suitable for most messages (including messages containing MQ header structures), but some messages might require more stringent alignment. For example, a message containing a 64-bit binary integer might require 8-byte alignment.

If *Buffer* contains character or numeric data, set the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter to the values appropriate to the data; this enables the receiver of the message to convert the data (if necessary) to the character set and encoding used by the receiver.

Note: All the other parameters on the MQPUT1 call must be in the character set and encoding of the local queue manager (given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE).

In the C programming language, the parameter is declared as a pointer-to-void; the address of any type of data can be specified as the parameter.

If the *BufferLength* parameter is zero, *Buffer* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler can be null.

CompCode (MQLONG) – output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason (MQLONG) – output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_MULTIPLE_REASONS
(2136, X'858') Multiple reason codes returned.

MQRC_INCOMPLETE_GROUP
(2241, X'8C1') Message group not complete.

MQRC_INCOMPLETE_MSG
(2242, X'8C2') Logical message not complete.

MQRC_PRIORITY_EXCEEDS_MAXIMUM
(2049, X'801') Message Priority exceeds maximum value supported.

MQRC_UNKNOWN_REPORT_OPTION
(2104, X'838') Report option(s) in message descriptor not recognized.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ALIAS_BASE_Q_TYPE_ERROR
(2001, X'7D1') Alias base queue not a valid type.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BACKED_OUT
(2003, X'7D3') Unit of work backed out.

MQRC_BUFFER_ERROR
(2004, X'7D4') Buffer parameter not valid.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_NOT_AVAILABLE
(2345, X'929') Coupling facility not available.

MQRC_CF_STRUC_AUTH_FAILED
(2348, X'92C') Coupling-facility structure authorization check failed.

MQRC_CF_STRUC_ERROR
(2349, X'92D') Coupling-facility structure not valid.

MQRC_CF_STRUC_FAILED
(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CF_STRUC_LIST_HDR_IN_USE
(2347, X'92B') Coupling-facility structure list-header in use.

MQRC_CFGR_ERROR
(2416, X'970') PCF group parameter structure MQCFGR in the message data is not valid.

MQRC_CFH_ERROR
(2235, X'8BB') PCF header structure not valid.

MQRC_CFIF_ERROR
(2414, X'96E') PCF integer filter parameter structure in the message data is not valid.

MQRC_CFIL_ERROR
(2236, X'8BC') PCF integer list parameter structure or PCIF*64 integer list parameter structure not valid.

MQRC_CFIN_ERROR
(2237, X'8BD') PCF integer parameter structure or PCIF*64 integer parameter structure not valid.

MQRC_CFSF_ERROR
(2415, X'96F') PCF string filter parameter structure in the message data is not valid.

MQRC_CFSL_ERROR
(2238, X'8BE') PCF string list parameter structure not valid.

MQRC_CFST_ERROR
(2239, X'8BF') PCF string parameter structure not valid.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CLUSTER_EXIT_ERROR
(2266, X'8DA') Cluster workload exit failed.

MQRC_CLUSTER_RESOLUTION_ERROR
(2189, X'88D') Cluster name resolution failed.

MQRC_CLUSTER_RESOURCE_ERROR
(2269, X'8DD') Cluster resource error.

MQRC_COD_NOT_VALID_FOR_XCF_Q
(2106, X'83A') COD report option not valid for XCF queue.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_CONTEXT_HANDLE_ERROR
(2097, X'831') Queue handle referred to does not save context.

MQRC_CONTEXT_NOT_AVAILABLE
(2098, X'832') Context not available for queue handle referred to.

MQRC_DATA_LENGTH_ERROR
(2010, X'7DA') Data length parameter not valid.

MQRC_DB2_NOT_AVAILABLE
(2342, X'926') DB2 subsystem not available.

MQRC_DEF_XMIT_Q_TYPE_ERROR
(2198, X'896') Default transmission queue not local.

MQRC_DEF_XMIT_Q_USAGE_ERROR
(2199, X'897') Default transmission queue usage error.

MQRC_DH_ERROR
(2135, X'857') Distribution header structure not valid.

MQRC_DLH_ERROR
(2141, X'85D') Dead letter header structure not valid.

MQRC_EPH_ERROR
(2420, X'974') Embedded PCF structure not valid.

MQRC_EXPIRY_ERROR
(2013, X'7DD') Expiry time not valid.

MQRC_FEEDBACK_ERROR
(2014, X'7DE') Feedback code not valid.

MQRC_GLOBAL_UOW_CONFLICT
(2351, X'92F') Global units of work conflict.

MQRC_GROUP_ID_ERROR
(2258, X'8D2') Group identifier not valid.

MQRC_HANDLE_IN_USE_FOR_UOW
(2353, X'931') Handle in use for global unit of work.

MQRC_HANDLE_NOT_AVAILABLE
(2017, X'7E1') No more handles available.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HEADER_ERROR
(2142, X'85E') MQ header structure not valid.

MQRC_IH_ERROR
(2148, X'864') IMS information header structure not valid.

MQRC_LOCAL_UOW_CONFLICT
(2352, X'930') Global unit of work conflicts with local unit of work.

MQRC_MD_ERROR
(2026, X'7EA') Message descriptor not valid.

MQRC_MDE_ERROR
(2248, X'8C8') Message descriptor extension not valid.

MQRC_MISSING_REPLY_TO_Q
(2027, X'7EB') Missing reply-to queue.

MQRC_MISSING_WIH
(2332, X'91C') Message data does not begin with MQWIH.

MQRC_MSG_FLAGS_ERROR
(2249, X'8C9') Message flags not valid.

MQRC_MSG_SEQ_NUMBER_ERROR
(2250, X'8CA') Message sequence number not valid.

MQRC_MSG_TOO_BIG_FOR_Q
(2030, X'7EE') Message length greater than maximum for queue.

MQRC_MSG_TOO_BIG_FOR_Q_MGR
(2031, X'7EF') Message length greater than maximum for queue manager.

MQRC_MSG_TYPE_ERROR
(2029, X'7ED') Message type in message descriptor not valid.

MQRC_MULTIPLE_REASONS
(2136, X'858') Multiple reason codes returned.

MQRC_NO_DESTINATIONS_AVAILABLE
(2270, X'8DE') No destination queues available.

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OBJECT_IN_USE
(2042, X'7FA') Object already open with conflicting options.

MQRC_OBJECT_LEVEL_INCOMPATIBLE
(2360, X'938') Object level not compatible.

MQRC_OBJECT_NAME_ERROR
(2152, X'868') Object name not valid.

MQRC_OBJECT_NOT_UNIQUE
(2343, X'927') Object not unique.

MQRC_OBJECT_Q_MGR_NAME_ERROR
(2153, X'869') Object queue-manager name not valid.

MQRC_OBJECT_RECORDS_ERROR
(2155, X'86B') Object records not valid.

MQRC_OBJECT_TYPE_ERROR
(2043, X'7FB') Object type not valid.

MQRC_OD_ERROR
(2044, X'7FC') Object descriptor structure not valid.

MQRC_OFFSET_ERROR
(2251, X'8CB') Message segment offset not valid.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_ORIGINAL_LENGTH_ERROR
(2252, X'8CC') Original length not valid.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_PAGESET_FULL
(2192, X'890') External storage medium is full.

MQRC_PCF_ERROR
(2149, X'865') PCF structures not valid.

MQRC_PERSISTENCE_ERROR
(2047, X'7FF') Persistence not valid.

MQRC_PERSISTENT_NOT_ALLOWED
(2048, X'800') Queue does not support persistent messages.

MQRC_PMO_ERROR
(2173, X'87D') Put-message options structure not valid.

MQRC_PMO_RECORD_FLAGS_ERROR
(2158, X'86E') Put message record flags not valid.

MQRC_PRIORITY_ERROR
(2050, X'802') Message priority not valid.

MQRC_PUT_INHIBITED
(2051, X'803') Put calls inhibited for the queue.

MQRC_PUT_MSG_RECORDS_ERROR
(2159, X'86F') Put message records not valid.

MQRC_Q_DELETED
(2052, X'804') Queue has been deleted.

MQRC_Q_FULL
(2053, X'805') Queue already contains maximum number of messages.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR QUIESCING
(2161, X'871') Queue manager quiescing.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_Q_SPACE_NOT_AVAILABLE
(2056, X'808') No space available on disk for queue.

MQRC_Q_TYPE_ERROR
(2057, X'809') Queue type not valid.

MQRC_RECS_PRESENT_ERROR
(2154, X'86A') Number of records present not valid.

MQRC_REMOTE_Q_NAME_ERROR
(2184, X'888') Remote queue name not valid.

MQRC_REPORT_OPTIONS_ERROR
(2061, X'80D') Report options in message descriptor not valid.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_RESPONSE_RECORDS_ERROR
(2156, X'86C') Response records not valid.

MQRC_RFH_ERROR
(2334, X'91E') MQRFH or MQRFH2 structure not valid.

MQRC_RMH_ERROR
(2220, X'8AC') Reference message header structure not valid.

MQRC_SECURITY_ERROR
(2063, X'80F') Security error occurred.

MQRC_SEGMENT_LENGTH_ZERO
(2253, X'8CD') Length of data in message segment is zero.

MQRC_STOPPED_BY_CLUSTER_EXIT
(2188, X'88C') Call rejected by cluster workload exit.

MQRC_STORAGE_CLASS_ERROR
(2105, X'839') Storage class error.

MQRC_STORAGE_MEDIUM_FULL
(2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_SYNCPOINT_LIMIT_REACHED
(2024, X'7E8') No more messages can be handled within current unit of work.

MQRC_SYNCPOINT_NOT_AVAILABLE
(2072, X'818') Syncpoint support not available.

MQRC_TM_ERROR
(2265, X'8D9') Trigger message structure not valid.

MQRC_TMC_ERROR
(2191, X'88F') Character trigger message structure not valid.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

MQRC_UNKNOWN_ALIAS_BASE_Q
(2082, X'822') Unknown alias base queue.

MQRC_UNKNOWN_DEF_XMIT_Q
(2197, X'895') Unknown default transmission queue.

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRC_UNKNOWN_OBJECT_Q_MGR
(2086, X'826') Unknown object queue manager.

MQRC_UNKNOWN_REMOTE_Q_MGR
(2087, X'827') Unknown remote queue manager.

MQRC_UNKNOWN_XMIT_Q
(2196, X'894') Unknown transmission queue.

MQRC_UOW_ENLISTMENT_ERROR
(2354, X'932') Enlistment in global unit of work failed.

MQRC_UOW_MIX_NOT_SUPPORTED
(2355, X'933') Mixture of unit-of-work calls not supported.

MQRC_UOW_NOT_AVAILABLE
(2255, X'8CF') Unit of work not available for the queue manager to use.

MQRC_WIH_ERROR
(2333, X'91D') MQWIH structure not valid.

MQRC_WRONG_CF_LEVEL
(2366, X'93E') Coupling-facility structure is wrong level.

MQRC_WRONG_MD_VERSION
(2257, X'8D1') Wrong version of MQMD supplied.

MQRC_XMIT_Q_TYPE_ERROR
(2091, X'82B') Transmission queue not local.

MQRC_XMIT_Q_USAGE_ERROR
(2092, X'82C') Transmission queue with wrong usage.

MQRC_XQH_ERROR
(2260, X'8D4') Transmission queue header structure not valid.

For detailed information on these codes, see:

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

Usage notes for MQPUT1

1. Both the MQPUT and MQPUT1 calls can be used to put messages on a queue; which call to use depends on the circumstances:
 - Use the MQPUT call to place multiple messages on the *same* queue.
An MQOPEN call specifying the MQOO_OUTPUT option is issued first, followed by one or more MQPUT requests to add messages to the queue; finally the queue is closed with an MQCLOSE call. This gives better performance than repeated use of the MQPUT1 call.
 - Use the MQPUT1 call to put only *one* message on a queue.
This call encapsulates the MQOPEN, MQPUT, and MQCLOSE calls into a single call, minimizing the number of calls that must be issued.
2. If an application puts a sequence of messages on the same queue without using message groups, the order of those messages is preserved provided that certain conditions are satisfied. However, in most environments the MQPUT1 call does not satisfy these conditions, and so does not preserve message order. The MQPUT call must be used instead in these environments. See MQPUT usage notes for details.
3. 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. Distribution lists are supported in the following environments: AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

The following differences apply when using the MQPUT1 call:

a. If the application provides MQRR response records, they must be provided using the MQOD structure; they cannot be provided using the MQPMO structure.

b. The reason code MQRC_OPEN_FAILED is never returned by MQPUT1 in the response records; if a queue fails to open, the response record for that queue contains the reason code resulting from the open operation.

If an open operation for a queue succeeds with a completion code of MQCC_WARNING, the completion code and reason code in the response record for that queue are replaced by the completion and reason codes resulting from the put operation.

As with the MQOPEN and MQPUT calls, the queue manager sets the response records (if provided) only when the outcome of the call is not the same for all queues in the distribution list; this is indicated by the call completing with reason code MQRC_MULTIPLE_REASONS.

4. If the MQPUT1 call is used to put a message on a cluster queue, the call behaves as though MQOO_BIND_NOT_FIXED had been specified on the MQOPEN call.

5. If a message is put with one or more MQ header structures at the beginning of the application message data, the queue manager performs certain checks on the header structures to verify that they are valid. For more information about this, see the usage notes for the MQPUT call.

6. If more than one of the warning situations arise (see the *CompCode* parameter), the reason code returned is the *first* one in the following list that applies:

a. MQRC_MULTIPLE_REASONS

b. MQRC_INCOMPLETE_MSG

c. MQRC_INCOMPLETE_GROUP

d. MQRC_PRIORITY_EXCEEDS_MAXIMUM or
MQRC_UNKNOWN_REPORT_OPTION

7. For the Visual Basic programming language, the following points apply:

- If the size of the *Buffer* parameter is less than the length specified by the *BufferLength* parameter, the call fails with reason code MQRC_BUFFER_LENGTH_ERROR.

- The *Buffer* parameter is declared as being of type *String*. If the data to be placed on the queue is not of type *String*, use the MQPUT1Any call in place of MQPUT1.

The MQPUT1Any call has the same parameters as the MQPUT1 call, except that the *Buffer* parameter is declared as being of type *Any*, allowing any type of data to be placed on the queue. However, this means that *Buffer* cannot be checked to ensure that it is at least *BufferLength* bytes in size.

Language invocations for MQPUT1

The MQPUT1 call is supported in the programming languages shown below.

C invocation

```
MQPUT1 (Hconn, &ObjDesc, &MsgDesc, &PutMsgOpts,  
        BufferLength, Buffer, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;          /* Connection handle */  
MQOD    ObjDesc;       /* Object descriptor */  
MQMD    MsgDesc;       /* Message descriptor */
```



```

MQPMO   PutMsgOpts;   /* Options that control the action of MQPUT1 */
MQLONG  BufferLength; /* Length of the message in Buffer */
MQBYTE  Buffer[n];    /* Message data */
MQLONG  CompCode;    /* Completion code */
MQLONG  Reason;      /* Reason code qualifying CompCode */

```

COBOL invocation

```

CALL 'MQPUT1' USING HCONN, OBJDESC, MSGDESC, PUTMSGOPTS,
                   BUFFERLENGTH, BUFFER, COMPCODE, REASON.

```

Declare the parameters as follows:

```

** Connection handle
01 HCONN          PIC S9(9) BINARY.
** Object descriptor
01 OBJDESC.
   COPY CMQODV.
** Message descriptor
01 MSGDESC.
   COPY CMQMDV.
** Options that control the action of MQPUT1
01 PUTMSGOPTS.
   COPY CMQPMOV.
** Length of the message in BUFFER
01 BUFFERLENGTH PIC S9(9) BINARY.
** Message data
01 BUFFER       PIC X(n).
** Completion code
01 COMPCODE     PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON      PIC S9(9) BINARY.

```

PL/I invocation

```

call MQPUT1 (Hconn, ObjDesc, MsgDesc, PutMsgOpts, BufferLength, Buffer,
            CompCode, Reason);

```

Declare the parameters as follows:

```

dcl Hconn          fixed bin(31); /* Connection handle */
dcl ObjDesc       like MQOD;     /* Object descriptor */
dcl MsgDesc       like MQMD;     /* Message descriptor */
dcl PutMsgOpts    like MQPMO;    /* Options that control the action of
                                MQPUT1 */
dcl BufferLength   fixed bin(31); /* Length of the message in Buffer */
dcl Buffer         char(n);       /* Message data */
dcl CompCode      fixed bin(31); /* Completion code */
dcl Reason        fixed bin(31); /* Reason code qualifying CompCode */

```

System/390 assembler invocation

```

CALL MQPUT1,(HCONN,OBJDESC,MSGDESC,PUTMSGOPTS,BUFFERLENGTH, X
            BUFFER,COMPCODE,REASON)

```

Declare the parameters as follows:

```

HCONN          DS      F      Connection handle
OBJDESC        CMQODA   ,      Object descriptor
MSGDESC        CMQMDA   ,      Message descriptor
PUTMSGOPTS     CMQPMOA  ,      Options that control the action of MQPUT1
BUFFERLENGTH   DS      F      Length of the message in BUFFER
BUFFER         DS      CL(n)  Message data
COMPCODE       DS      F      Completion code
REASON         DS      F      Reason code qualifying COMPCODE

```

Visual Basic invocation

```

MQPUT1 Hconn, ObjDesc, MsgDesc, PutMsgOpts, BufferLength, Buffer,
      CompCode, Reason

```

Declare the parameters as follows:

Dim Hconn	As Long	'Connection handle'
Dim ObjDesc	As MQOD	'Object descriptor'
Dim MsgDesc	As MQMD	'Message descriptor'
Dim PutMsgOpts	As MQPMO	'Options that control the action of MQPUT1'
Dim BufferLength	As Long	'Length of the message in Buffer'
Dim Buffer	As String	'Message data'
Dim CompCode	As Long	'Completion code'
Dim Reason	As Long	'Reason code qualifying CompCode'

MQSET – Set object attributes

Use the MQSET call to change the attributes of an object represented by a handle. The object must be a queue.

Syntax for MQSET

MQSET (*Hconn*, *Hobj*, *SelectorCount*, *Selectors*, *IntAttrCount*, *IntAttrs*, *CharAttrLength*, *CharAttrs*, *CompCode*, *Reason*)

Parameters for MQSET

The MQSET call has the following parameters.

Hconn (MQHCONN) – input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on i5/OS for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN

Default connection handle.

Hobj (MQHOBJ) – input

This handle represents the queue object whose attributes are to be set. The handle was returned by a previous MQOPEN call that specified the MQOO_SET option.

SelectorCount (MQLONG) – input

This is the count of selectors that are supplied in the *Selectors* array. It is the number of attributes that are to be set. Zero is a valid value. The maximum number allowed is 256.

Selectors (MQLONGxSelectorCount) – input

This is an array of *SelectorCount* attribute selectors; each selector identifies an attribute (integer or character) whose value is to be set.

Each selector must be valid for the type of queue that *Hobj* represents. Only certain MQIA_* and MQCA_* values are allowed; these values are listed below.

Selectors can be specified in any order. Attribute values that correspond to integer attribute selectors (MQIA_* selectors) must be specified in *IntAttrs* in the same order in which these selectors occur in *Selectors*. Attribute values that correspond

to character attribute selectors (MQCA_* selectors) must be specified in *CharAttrs* in the same order in which those selectors occur. MQIA_* selectors can be interleaved with the MQCA_* selectors; only the relative order within each type is important.

You can specify the same selector more than once; if you do, the last value specified for a given selector is the one that takes effect.

Note:

1. The integer and character attribute selectors are allocated within two different ranges; the MQIA_* selectors reside within the range MQIA_FIRST through MQIA_LAST, and the MQCA_* selectors within the range MQCA_FIRST through MQCA_LAST.

For each range, the constants MQIA_LAST_USED and MQCA_LAST_USED define the highest value that the queue manager accepts.

2. If all the MQIA_* selectors occur first, the same element numbers can be used to address corresponding elements in the *Selectors* and *IntAttrs* arrays.
3. If the *SelectorCount* parameter is zero, *Selectors* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler might be null.

The attributes that can be set are listed in the following table. No other attributes can be set using this call. For the MQCA_* attribute selectors, the constant that defines the length in bytes of the string that is required in *CharAttrs* is given in parentheses.

Table 94. MQSET attribute selectors for queues

Selector	Description	Note
MQCA_TRIGGER_DATA	Trigger data (MQ_TRIGGER_DATA_LENGTH).	
MQIA_DIST_LISTS	Distribution list support.	1
MQIA_INHIBIT_GET	Whether get operations are allowed.	
MQIA_INHIBIT_PUT	Whether put operations are allowed.	
MQIA_TRIGGER_CONTROL	Trigger control.	
MQIA_TRIGGER_DEPTH	Trigger depth.	
MQIA_TRIGGER_MSG_PRIORITY	Threshold message priority for triggers.	
MQIA_TRIGGER_TYPE	Trigger type.	
Note:		
1. Supported only on AIX, HP-UX, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.		

IntAttrCount (MQLONG) – input

This is the number of elements in the *IntAttrs* array, and must be at least the number of MQIA_* selectors in the *Selectors* parameter. Zero is a valid value if there are none.

IntAttrs (MQLONGxIntAttrCount) – input

This is an array of *IntAttrCount* integer attribute values. These attribute values must be in the same order as the MQIA_* selectors in the *Selectors* array.

If the *IntAttrCount* or *SelectorCount* parameter is zero, *IntAttrs* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler might be null.

CharAttrLength (MQLONG) – input

This is the length in bytes of the *CharAttrs* parameter, and must be at least the sum of the lengths of the character attributes specified in the *Selectors* array. Zero is a valid value if there are no MQCA_* selectors in *Selectors*.

CharAttrs (MQCHARxCharAttrLength) – input

This is the buffer containing the character attribute values, concatenated together. The length of the buffer is given by the *CharAttrLength* parameter.

The characters attributes must be specified in the same order as the MQCA_* selectors in the *Selectors* array. The length of each character attribute is fixed (see *Selectors*). If the value to be set for an attribute contains fewer nonblank characters than the defined length of the attribute, pad the value in *CharAttrs* to the right with blanks to make the attribute value match the defined length of the attribute.

If the *CharAttrLength* or *SelectorCount* parameter is zero, *CharAttrs* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler might be null.

CompCode (MQLONG) – output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason (MQLONG) – output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_STRUC_FAILED
(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CF_STRUC_LIST_HDR_IN_USE
(2347, X'92B') Coupling-facility structure list-header in use.

MQRC_CHAR_ATTR_LENGTH_ERROR
(2006, X'7D6') Length of character attributes not valid.

MQRC_CHAR_ATTRS_ERROR
(2007, X'7D7') Character attributes string not valid.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_DB2_NOT_AVAILABLE
(2342, X'926') DB2 subsystem not available.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_INHIBIT_VALUE_ERROR
(2020, X'7E4') Value for inhibit-get or inhibit-put queue attribute not valid.

MQRC_INT_ATTR_COUNT_ERROR
(2021, X'7E5') Count of integer attributes not valid.

MQRC_INT_ATTRS_ARRAY_ERROR
(2023, X'7E7') Integer attributes array not valid.

MQRC_NOT_OPEN_FOR_SET
(2040, X'7F8') Queue not open for set.

MQRC_OBJECT_CHANGED
(2041, X'7F9') Object definition changed since opened.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_Q_DELETED
(2052, X'804') Queue has been deleted.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

- MQRC_Q_MGR_NOT_AVAILABLE**
(2059, X'80B') Queue manager not available for connection.
- MQRC_Q_MGR_STOPPING**
(2162, X'872') Queue manager shutting down.
- MQRC_RESOURCE_PROBLEM**
(2102, X'836') Insufficient system resources available.
- MQRC_SELECTOR_COUNT_ERROR**
(2065, X'811') Count of selectors not valid.
- MQRC_SELECTOR_ERROR**
(2067, X'813') Attribute selector not valid.
- MQRC_SELECTOR_LIMIT_EXCEEDED**
(2066, X'812') Count of selectors too big.
- MQRC_STORAGE_NOT_AVAILABLE**
(2071, X'817') Insufficient storage available.
- MQRC_SUPPRESSED_BY_EXIT**
(2109, X'83D') Call suppressed by exit program.
- MQRC_TRIGGER_CONTROL_ERROR**
(2075, X'81B') Value for trigger-control attribute not valid.
- MQRC_TRIGGER_DEPTH_ERROR**
(2076, X'81C') Value for trigger-depth attribute not valid.
- MQRC_TRIGGER_MSG_PRIORITY_ERR**
(2077, X'81D') Value for trigger-message-priority attribute not valid.
- MQRC_TRIGGER_TYPE_ERROR**
(2078, X'81E') Value for trigger-type attribute not valid.
- MQRC_UNEXPECTED_ERROR**
(2195, X'893') Unexpected error occurred.

For detailed information on these codes, see:

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

Usage notes for MQSET

1. Using this call, the application can specify an array of integer attributes, or a collection of character attribute strings, or both. If no errors occur, the attributes specified are all set simultaneously. If an error occurs (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 object attributes” on page 480 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. You cannot change the attributes of a model queue using the MQSET call. However, if you open a model queue using the MQOPEN call with the

MQOO_SET option, you can use the MQSET call to set the attributes of the dynamic local queue that is created by the MQOPEN call.

5. If the object being set is a cluster queue, there must be a local instance of the cluster queue for the open to succeed.
6. Changes to attributes resulting from use of the MQSET call do not affect the values of the *AlterationDate* and *AlterationTime* attributes.
7. For more information about object attributes, see:
 - “Attributes for queues” on page 615
 - “Attributes for namelists” on page 651
 - “Attributes for process definitions” on page 654
 - “Attributes for the queue manager” on page 575

Language invocations for MQSET

The MQSET call is supported in the programming languages shown below.

C invocation

```
MQSET (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,  
      CharAttrLength, CharAttrs, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;          /* Connection handle */  
MQHOBJ   Hobj;          /* Object handle */  
MQLONG   SelectorCount; /* Count of selectors */  
MQLONG   Selectors[n];  /* Array of attribute selectors */  
MQLONG   IntAttrCount;  /* Count of integer attributes */  
MQLONG   IntAttrs[n];   /* Array of integer attributes */  
MQLONG   CharAttrLength; /* Length of character attributes buffer */  
MQCHAR   CharAttrs[n];  /* Character attributes */  
MQLONG   CompCode;      /* Completion code */  
MQLONG   Reason;       /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQSET' USING HCONN, HOBJ, SELECTORCOUNT, SELECTORS-TABLE,  
                  INTATTRCOUNT, INTATTRS-TABLE, CHARATTRLENGTH,  
                  CHARATTRS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle  
01 HCONN          PIC S9(9) BINARY.  
** Object handle  
01 HOBJ          PIC S9(9) BINARY.  
** Count of selectors  
01 SELECTORCOUNT PIC S9(9) BINARY.  
** Array of attribute selectors  
01 SELECTORS-TABLE.  
02 SELECTORS     PIC S9(9) BINARY OCCURS n TIMES.  
** Count of integer attributes  
01 INTATTRCOUNT PIC S9(9) BINARY.  
** Array of integer attributes  
01 INTATTRS-TABLE.  
02 INTATTRS     PIC S9(9) BINARY OCCURS n TIMES.  
** Length of character attributes buffer  
01 CHARATTRLENGTH PIC S9(9) BINARY.  
** Character attributes  
01 CHARATTRS     PIC X(n).  
** Completion code  
01 COMPCODE      PIC S9(9) BINARY.  
** Reason code qualifying COMPCODE  
01 REASON        PIC S9(9) BINARY.
```


PL/I invocation

```
call MQSET (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount,  
            IntAttrs, CharAttrLength, CharAttrs, CompCode, Reason);
```

Declare the parameters as follows:

```
dc1 Hconn          fixed bin(31); /* Connection handle */  
dc1 Hobj           fixed bin(31); /* Object handle */  
dc1 SelectorCount  fixed bin(31); /* Count of selectors */  
dc1 Selectors(n)   fixed bin(31); /* Array of attribute selectors */  
dc1 IntAttrCount   fixed bin(31); /* Count of integer attributes */  
dc1 IntAttrs(n)    fixed bin(31); /* Array of integer attributes */  
dc1 CharAttrLength fixed bin(31); /* Length of character attributes  
                                buffer */  
  
dc1 CharAttrs      char(n);      /* Character attributes */  
dc1 CompCode       fixed bin(31); /* Completion code */  
dc1 Reason         fixed bin(31); /* Reason code qualifying  
                                CompCode */
```

System/390 assembler invocation

```
CALL MQSET, (HCONN,HOBJ,SELECTORCOUNT,SELECTORS,INTATTRCOUNT, X  
            INTATTRS,CHARATTRLENGTH,CHARATTRS,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN          DS  F      Connection handle  
HOBJ           DS  F      Object handle  
SELECTORCOUNT DS  F      Count of selectors  
SELECTORS      DS  (n)F   Array of attribute selectors  
INTATTRCOUNT  DS  F      Count of integer attributes  
INTATTRS       DS  (n)F   Array of integer attributes  
CHARATTRLENGTH DS  F      Length of character attributes buffer  
CHARATTRS      DS  CL(n)  Character attributes  
COMPCODE       DS  F      Completion code  
REASON         DS  F      Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQSET Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,  
      CharAttrLength, CharAttrs, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn          As Long 'Connection handle'  
Dim Hobj           As Long 'Object handle'  
Dim SelectorCount  As Long 'Count of selectors'  
Dim Selectors      As Long 'Array of attribute selectors'  
Dim IntAttrCount  As Long 'Count of integer attributes'  
Dim IntAttrs       As Long 'Array of integer attributes'  
Dim CharAttrLength As Long 'Length of character attributes buffer'  
Dim CharAttrs      As String 'Character attributes'  
Dim CompCode       As Long 'Completion code'  
Dim Reason         As Long 'Reason code qualifying CompCode'
```

MQSETMP – Set message property

Call that sets a property of a message handle.

The MQSETMP call sets or modifies a property of a message handle.

Syntax for MQSETMP

MQSETMP call syntax and list of parameters

MQSETMP (*Hconn, Hmsg, SetPropOpts, Name, PropDesc, Type, ValueLength, Value, CompCode, Reason*)

Parameters for MQSETMP

List of valid parameters for the MQSETMP call.

The MQSETMP call has the following parameters:

Hconn (MQHCONN) – input

This handle represents the connection to the queue manager.

The value must match the connection handle that was used to create the message handle specified in the *Hmsg* parameter.

If the message handle was created using MQHC_UNASSOCIATED_HCONN, a valid connection must be established on the thread setting a property of the message handle, otherwise the call fails with reason code MQRC_CONNECTION_BROKEN.

Hmsg (MQHMSG) – input

This is the message handle to be modified. The value was returned by a previous MQCRTMH call.

SetPropsOpts (MQSMPO) – input

Control how message properties are set.

This structure allows applications to specify options that control how message properties are set. The structure is an input parameter on the MQSETMP call. See MQSMPO for further information.

Name (MQCHARV) – input

This is the name of the property to set.

See the *WebSphere MQ Application Programming Guide* for further information about the use of property names.

PropDesc (MQPD) – input/output

This structure is used to define the attributes of a property, including:

- what happens if the property is not supported
- what message context the property belongs to
- what messages the property is copied into as it flows

See MQPD for further information about this structure.

Type (MQLONG) – input

The data type of the property being set. It can be one of the following:

MQTYPE_BOOLEAN

A boolean. *ValueLength* must be 4.

MQTYPE_BYTE_STRING

A byte string. *ValueLength* must be zero or greater.

| **MQTYPE_INT8**

| An 8-bit signed integer. *ValueLength* must be 1.

| **MQTYPE_INT16**

| A 16-bit signed integer. *ValueLength* must be 2.

| **MQTYPE_INT32**

| A 32-bit signed integer. *ValueLength* must be 4.

| **MQTYPE_INT64**

| A 64-bit signed integer. *ValueLength* must be 8.

| **MQTYPE_FLOAT32**

| A 32-bit floating-point number. *ValueLength* must be 4.

| Note: this type is not supported with applications using IBM COBOL for
| z/OS.

| **MQTYPE_FLOAT64**

| A 64-bit floating-point number. *ValueLength* must be 8.

| Note: this type is not supported with applications using IBM COBOL for
| z/OS.

| **MQTYPE_STRING**

| A character string. *ValueLength* must be zero or greater, or the special value
| MQVL_NULL_TERMINATED.

| **MQTYPE_NULL**

| The property exists but has a null value. *ValueLength* must be zero.

| **ValueLength (MQLONG) – input**

| The length in bytes of the property value in the *Value* parameter. Zero is valid only
| for null values or for strings or byte strings. Zero indicates that the property exists
| but that the value contains no characters or bytes.

| The value must be greater than or equal to zero or the following special value if
| the *Type* parameter has MQTYPE_STRING set:

| **MQVL_NULL_TERMINATED**

| The value is delimited by the first null encountered in the string. The null
| is not included as part of the string. This value is invalid if
| MQTYPE_STRING is not also set.

| Note: The null character used to terminate a string if
| MQVL_NULL_TERMINATED is set is a null from the character set of the
| Value.

| **Value (MQBYTE x ValueLength) – input**

| The value of the property to be set. The buffer must be aligned on a boundary
| appropriate to the nature of the data in the value.

| In the C programming language, the parameter is declared as a pointer-to-void; the
| address of any type of data can be specified as the parameter.

| If *ValueLength* is zero, *Value* is not referred to. In this case, the parameter address
| passed by programs written in C or System/390 assembler can be null.

| **CompCode (MQLONG) – output**

| The completion code; it is one of the following:

| MQCC_OK
| Successful completion.

| MQCC_FAILED
| Call failed.

| Reason (MQLONG) – output

| The reason code qualifying *CompCode*.

| If *CompCode* is MQCC_OK:

| MQRC_NONE
| (0, X'000') No reason to report.

| If *CompCode* is MQCC_WARNING:

| MQRC_RFH_FORMAT_ERROR
| (2421, X'0975') An MQRFH2 folder containing properties could not be
| parsed.

| If *CompCode* is MQCC_FAILED:

| MQRC_ADAPTER_NOT_AVAILABLE
| (2204, X'089C') Adapter not available.

| MQRC_ADAPTER_SERV_LOAD_ERROR
| (2130, X'852') Unable to load adapter service module.

| MQRC_ASID_MISMATCH
| (2157, X'86D') Primary and home ASIDs differ.

| MQRC_BUFFER_ERROR
| (2004, X'07D4') Value parameter not valid.

| MQRC_BUFFER_LENGTH_ERROR
| (2005, X'07D5') Value length parameter not valid.

| MQRC_CALL_IN_PROGRESS
| (2219, X'08AB') MQI call entered before previous call completed.

| MQRC_HMSG_ERROR
| (2460, X'099C') Message handle pointer not valid.

| MQRC_MSG_HANDLE_IN_USE
| (2499, X'09C3') Message handle already in use.

| MQRC_OPTIONS_ERROR
| (2046, X'07FE') Options not valid or not consistent.

| MQRC_PD_ERROR
| (2482, X'09B2') Property descriptor structure not valid.

| MQRC_PROPERTY_NAME_ERROR
| (2442, X'098A') Invalid property name.

| MQRC_PROPERTY_TYPE_ERROR
| (2473, X'09A9') Invalid property data type.

| MQRC_PROP_NUMBER_FORMAT_ERROR
| (2472, X'09A8') Number format error encountered in value data.

| MQRC_SMPO_ERROR
| (2463, X'099F') Set message property options structure not valid.

MQRC_SOURCE_CCSID_ERROR
 (2111, X'083F') Property name coded character set identifier not valid.

MQRC_STORAGE_NOT_AVAILABLE
 (2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
 (2195, X'893') Unexpected error occurred.

See Chapter 4, "Return codes," on page 661 for more details.

Language invocations for MQSETMP

The MQSETMP call is supported in the programming languages shown below. List of languages supporting the MQSETMP call

C invocation

Parameters used for the C invocation of MQSETMP.

```
MQSETMP (Hconn, Hmsg, &SetPropOpts, &Name, &PropDesc, Type,
ValueLength, &Value, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;      /* Connection handle */
MQHMSG  Hmsg;       /* Message handle */
MQSMPO  SetPropOpts; /* Options that control the action of MQSETMP */
MQCHARV Name;       /* Property name */
MQPD    PropDesc;   /* Property descriptor */
MQLONG  Type;       /* Property data type */
MQLONG  ValueLength; /* Length of property value in Value */
MQBYTE  Value[n];   /* Property value */
MQLONG  CompCode;   /* Completion code */
MQLONG  Reason;     /* Reason code qualifying CompCode */
```

COBOL invocation

Parameters used for the COBOL invocation of MQSETMP.

```
CALL 'MQSETMP' USING HCONN, HMSG, SETMSGOPTS, NAME, PROPDSC, TYPE,
VALUELENGTH, VALUE, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN    PIC S9(9) BINARY.
** Message handle
01 HMSG     PIC S9(18) BINARY.
** Options that control the action of MQSETMP
01 SETMSGOPTS.
   COPY CMQSMPOV.
** Property name
01 NAME
   COPY CMQCHRVV.
** Property descriptor
01 PROPDSC.
   COPY CMQPDV.
** Property data type
01 TYPE     PIC S9(9) BINARY.
** Length of property value in VALUE
01 VALUELENGTH PIC S9(9) BINARY.
** Property value
01 VALUE     PIC X(n).
** Completion code
01 COMPCODE  PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON    PIC S9(9) BINARY.
```

PL/I invocation

Parameters used for the PL/I invocation of MQSETMP.

```
call MQSETMP (Hconn, Hmsg, SetPropOpts, Name, PropDesc, Type, ValueLength,  
             Value, CompCode, Reason);
```

Declare the parameters as follows:

```
dc1 Hconn      fixed bin(31); /* Connection handle */  
dc1 Hmsg      fixed bin(63); /* Message handle */  
dc1 SetPropOpts like MQSMPO; /* Options that control the action of MQSETMP */  
dc1 Name       like MQCHARV; /* Property name */  
dc1 PropDesc   like MQPD;    /* Property descriptor */  
dc1 Type       fixed bin(31); /* Property data type */  
dc1 ValueLength fixed bin(31); /* Length of property value in Value */  
dc1 Value      char(n);      /* Property value */  
dc1 CompCode   fixed bin(31); /* Completion code */  
dc1 Reason     fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 assembler invocation

Parameters used for the System/390 assembler invocation of MQSETMP.

```
CALL MQSETMP, (HCONN,HMSG,SETMSGHOPTS,NAME,PROPDSC,TYPE,VALUELENGTH,  
              VALUE,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN      DS      F      Connection handle  
HMSG       DS      D      Message handle  
SETMSGOPTS CMQSMPOA ,      Options that control the action of MQSETMP  
NAME       CMQCHRVA ,      Property name  
PROPDSC    CMQPDA   ,      Property descriptor  
TYPE       DS      F      Property data type  
VALUELENGTH DS      F      Length of property value in VALUE  
VALUE      DS      CL(n) Property value  
COMPCODE   DS      F      Completion code  
REASON     DS      F      Reason code qualifying COMPCODE
```

MQSTAT – Retrieve status information

Use the MQSTAT call to retrieve status information. The type of status information returned is determined by the Type value specified on the call.

Syntax for MQSTAT

```
MQSTAT (Hconn, Type, Stat, CompCode, Reason)
```

Parameters for MQSTAT

The MQSTAT call has the following parameters.

Hconn (MQHCONN) – input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on i5/OS for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

```
MQHC_DEF_HCONN
```

Default connection handle.

Type (MQLONG) – input

Type of status information being requested. The only valid value is:

MQSTAT_TYPE_ASYNC_ERROR

Return information about previous asynchronous put operations.

Stat (MQSTS) – input/output

Status information structure. See “MQSTS – Status reporting structure” on page 361 for details.

CompCode (MQLONG) – output

The completion code; it is one of the following:

MQCC_OK

Successful completion.

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_API_EXIT_ERROR

(2374, X'946') API exit failed

MQRC_API_EXIT_LOAD_ERROR

(2183, X'887') Unable to load API exit.

MQRC_CALL_IN_PROGRESS

(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CONNECTION_BROKEN

(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_STOPPING

(2203, X'89B') Connection shutting down.

MQRC_HCONN_ERROR

(2018, X'7E2') Connection handle not valid.

MQRC_Q_MGR_STOPPING

(2162, X'872') Queue manager stopping

MQRC_RESOURCE_PROBLEM

(2102, X'836') Insufficient system resources available.

MQRC_STAT_TYPE_ERROR

(2430, X'97E') Error with MQSTAT type

MQRC_STORAGE_NOT_AVAILABLE

(2071, X'817') Insufficient storage available.

MQRC_STS_ERROR

(2424, X'978') Error with MQSTS structure

MQRC_UNEXPECTED_ERROR

(2195, X'893') Unexpected error occurred.

For detailed information on these codes, see:

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

Usage notes for MQSTAT

1. A call to MQSTAT specifying a type of MQSTAT_TYPE_ASYNC_ERROR returns information about previous asynchronous MQPUT and MQPUT1 operations. The MQSTS structure passed back on return from the MQSTAT call contains the first recorded asynchronous warning or error information for that connection. If further errors or warnings follow the first, they do not normally alter these values. However, if an error occurs with a completion code of MQCC_WARNING, a subsequent failure with a completion code of MQCC_FAILED is returned instead.
2. If no errors have occurred since the connection was established or since the last call to MQSTAT then a CompCode of MQCC_OK and Reason of MQRC_NONE are returned in the MQSTS structure.
3. Counts of the number of asynchronous calls that have been processed under the connection handle are returned via three counter fields; PutSuccessCount, PutWarningCount & PutFailureCount. These counters are incremented by the queue manager each time an asynchronous operation is processed successfully, has a warning or fails, respectively (note that for accounting purposes a put to a distribution list counts once per destination queue rather than once per distribution list). A counter will not be incremented beyond the maximum positive value AMQ_LONG_MAX.
4. A successful call to MQSTAT results in any previous error information or counts being reset.

Language invocations for MQSTAT

The MQSTAT call is supported in the programming languages shown below.

C invocation

```
MQSTAT (Hconn, StatType, &Stat, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn; /* Connection Handle */
MQLONG StatType; /* Status type */
MQSTS Stat; /* Status information structure */
MQLONG CompCode; /* Completion code */
MQLONG Reason; /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQSTAT' USING HCONN, STATTYPE, STAT, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN PIC S9(9) BINARY.
** Status type
01 STATTYPE PIC S9(9) BINARY.
** Status information
```

```

|         01 STAT.
|         COPY CMQSTSV.
|         ** Completion code
|         01 COMPCODE PIC S9(9) BINARY.
|         ** Reason code qualifying COMPCODE
|         01 REASON PIC S9(9) BINARY.

```

PL/I invocation (z/OS only)

```
call MQSTAT (Hconn, StatType, Stat, Compcode, Reason);
```

Declare the parameters as follows:

```

| dcl Hconn fixed bin(31); /* Connection handle */
| dcl StatType fixed bin(31); /* Status type */
| dcl Stat like MQSTS; /* Status information structure */
| dcl CompCode fixed bin(31); /* Completion code */
| dcl Reason fixed bin(31); /* Reason code qualifying CompCode */

```

System/390 assembler invocation

```
CALL MQSTAT,(HCONN,STATTYPE,STAT,COMPCODE,REASON)
```

Declare the parameters as follows:

```

| HCONN      DS      F      Connection handle
| STATTYPE   DS      F      Status type
| STAT       CMQSTSA,  Status information structure
| COMPCODE   DS      F      Completion code
| REASON     DS      F      Reason code qualifying COMPCODE

```

MQSUB - Register subscription

The MQSUB call registers the applications subscription to a particular topic.

Syntax for MQSUB

```
MQSUB (Hconn, SubDesc, Hobj, Hsub, CompCode, Reason)
```

Parameters for MQSUB

The MQSUB call has the following parameters.

Hconn (MQHCONN) - Input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on i5/OS for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

```
MQHC_DEF_HCONN
    Default connection handle.
```

SubDesc (MQSD) - input/output

This is a structure that identifies the object whose use is being registered by the application. See "MQSD - Subscription descriptor" on page 333 for more information.

Hobj (MQHOBJ) - Input/output

This handle represents the access that has been established to obtain the messages sent to this subscription. These messages can either be stored on a specific queue or the queue manager can manage their storage without using a specific queue.

To use a specific queue, you must associate it with the subscription when the subscription is created. You can do this in two ways:

- By using the DEFINE SUB MQSC command and providing that command with the name of a queue object.
- By providing this handle when calling MQSUB with the MQSO_CREATE

If this handle is provided as an input parameter on the call, it must be a valid object handle returned from a previous MQOPEN call of a queue using at least one of the following options:

- MQOO_INPUT_*
- MQOO_BROWSE
- MQOO_OUTPUT (if the queue is a remote queue)

If this is not the case, the call fails with MQRC_HOBJ_ERROR. It cannot be an object handle to an alias queue that resolves to a topic object. If this is the case, the call fails with MQRC_HOBJ_ERROR.

If the queue manager is to manage the storage of messages sent to this subscription, this should be set when you create the subscription, by using the MQSO_MANAGED option. The queue manager will then return this handle as an output parameter on the call. The handle that is returned is known as a managed handle. If MQHO_NONE is specified but MQSO_MANAGED is not specified, the call fails with MQRC_HOBJ_ERROR.

When a managed handle is returned to you by the queue manager, you can use it on an MQGET or MQCB call with or without browse options, on an MQINQ call, or on MQCLOSE. You cannot use it on MQPUT, MQSUB, MQSET; attempting to do so fails with MQRC_NOT_OPEN_FOR_OUTPUT, MQRC_HOBJ_ERROR, or MQRC_NOT_OPEN_FOR_SET respectively.

If this subscription is being resumed using the MQSO_RESUME option in the MQSD structure, the handle can be returned to the application in this parameter by setting MQSO_MANAGED to MQHO_NONE. You can do this whether the subscription is using a managed handle or not and may be useful to provide subscriptions created using DEFINE SUB with the handle to the subscription queue defined on that command. In the case where an administratively created subscription is being resumed, the queue opens with MQOO_INPUT_AS_Q_DEF and MQOO_BROWSE. If you need to specify other options, the application must open the subscription queue explicitly and provide the object handle on the call. If there is a problem opening the queue the call fails with MQRC_INVALID_DESTINATION. If the *Hobj* is provided, it must be equivalent to the *Hobj* in the original MQSUB call. This means if an object handle returned from an MQOPEN call is being provided, the handle must be to the same queue as previously used. If it is not the same queue, the call fails with MQRC_HOBJ_ERROR.

If this subscription is being altered using the MQSO_ALTER option in the MQSD structure, then a different *Hobj* can be provided. Any publications that have been delivered to the queue and were previously identified through this parameter stay

on that queue and it is the responsibility of the application to retrieve those messages if the *Hobj* parameter now represents a different queue.

The table summarises the use of this parameter with various subscription options:

Options	<i>Hobj</i>	Description
MQSO_CREATE + MQSO_MANAGED	Ignored on input	Creates a subscription with storage of messages managed by the queue manager
MQSO_CREATE	A valid object handle	Creates a subscription providing a specific queue as the destination for messages.
MQSO_RESUME	MQHO_NONE	Resumes a previously created subscription whether it was managed or not, and has the queue manager return the object handle for use by the application.
MQSO_RESUME	A valid, matching, object handle	Resumes a previously created subscription that uses a specific queue as the destination for messages and use an object handle with specific open options.
MQSO_ALTER + MQSO_MANAGED	MQHO_NONE	Alters an existing subscription that was previously using a specific queue, so it is now a managed subscription. The class of destination (managed or not) cannot be changed.
MQSO_ALTER	A valid object handle	Alters an existing subscription, whether it was managed or not, so that it now uses a specific queue. When the MQSO_MANAGED option is not used, the queue provided can be changed, but the class of destination (managed or not) cannot be changed.

Whether it was provided or returned, *Hobj* must be specified on subsequent MQGET or MQCB calls that want to receive the publication messages sent to this subscription.

The *Hobj* handle is no longer valid when the MQCLOSE call is issued on it, or when the unit of processing that defines the scope of the handle terminates. The scope of the object handle returned is the same as that of the connection handle specified on the call. See “Hconn (MQHCONN) – output” on page 438 for information about handle scope. An MQCLOSE of the *Hobj* handle has no effect on the *Hsub* handle.

Hsub (MQHOBJ) - output

This handle represents the subscription that has been made. It can be used for two further operations:

- It can be used on a subsequent MQSUBRQ call to request publications be sent when the MQSO_PUBLICATIONS_ON_REQUEST option has been used when making the subscription.

- It can be used on a subsequent MQCLOSE call to remove the subscription that has been made. The *Hsub* handle ceases to be valid when the MQCLOSE call is issued, or when the unit of processing that defines the scope of the handle terminates. The scope of the object handle returned is the same as that of the connection handle specified on the call. An MQCLOSE of the *Hsub* handle has no effect on the *Hobj* handle.

This handle cannot be passed to an MQGET or MQCB call. You must use the *Hobj* parameter. You cannot use this handle on any MQ call other than MQCLOSE or MQSUB. Passing this handle to any other MQ call results in MQRC_HOBJ_ERROR.

CompCode (MQLONG) - output

The completion code; it is one of the following:

- MQCC_OK**
Successful completion
- MQCC_WARNING**
Warning (partial completion)
- MQCC_FAILED**
Call failed

Reason (MQLONG) - output

The reason code qualifying *CompCode*

The reason code can be as follows:

If *CompCode* is MQCC_OK, the reason code is as follows:

- MQRC_NONE**
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED, the reason code is one of the following:

- MQRC_CLUSTER_RESOLUTION_ERROR**
(2189, X'88D') Cluster name resolution failed.
- MQRC_FUNCTION_NOT_SUPPORTED**
2298 (X'08FA') The function requested is not available in the current environment.
- MQRC_OPTIONS_ERROR**
2046 (X'07FE') Options parameter or field contains options that are not valid, or a combination of options that is not valid.
- MQRC_HOBJ_ERROR**
2019 (X'07E3') Object handle Hobj not valid
- MQRC_Q_MGR QUIESCING**
2161 (X'0871') Queue manager quiescing
- MQRC_UNKNOWN_OBJECT_NAME**
2085 (X'0825') Object identified cannot be found
- MQRC_SD_ERROR**
2424 (X'0978') Subscription descriptor (MQSD) not valid
- MQRC_SUB_NAME_ERROR**
2440 (X'0988') SubName field not valid

MQRC_OBJECT_STRING_ERROR
 2441 (X'0989') Objectstring field not valid

MQRC_TOPIC_STRING_ERROR
 2425 (X'0979') Topic string is not valid

MQRC_SUB_USER_DATA_ERROR
 2431 (X'097F') SubUserData field not valid

MQRC_SUB_ALREADY_EXISTS
 2432 (X'0980') Subscription already exists

MQRC_IDENTITY_MISMATCH
 2434 (X'0982') Subscription name matches existing subscription

MQRC_DURABILITY_NOT_ALLOWED
 2436 (X'0984') An MQSUB call using the MQSO_DURABLE option failed

Usage notes for MQSUB

1. The subscription is made to a topic, named either using the short name of a pre-defined topic object, the full name of the topic string, or it is formed by the concatenation of two parts. See the description of *ObjectName* and *ObjectString* in "MQSD - Subscription descriptor" on page 333.
2. The queue manager performs security checks when an MQSUB 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 appropriate topic object is located in the topic hierarchy and an authority check is made on this topic object to ensure authority to subscribe is set. If the MQSO_MANAGED option is not used, an authority check is made on the destination queue to ensure authority for output is set. If the MQSO_MANAGED option is used, no authority check is made on the managed queue for output or inquire access.
3. The Hobj returned on the MQSUB call when the MQSO_MANAGED option is used, can be inquired in order to find out attributes such as the Backout threshold and the Excessive backout requeue name. You can also inquire the name of the managed queue, but you should not attempt to directly open this queue.
4. Subscriptions can be grouped together allowing only a single publication to be delivered to the group of subscriptions even where more than one of the group matched the publication. Subscriptions are grouped using the MQSO_GROUP_SUB option and in order to group subscriptions together they must
 - be using the same named queue (that is not using the MQSO_MANAGED option) on the same queue manager – represented by the Hobj parameter on the MQSUB call
 - share the same SubCorrelId
 - be of the same SubLevel

These attributes define the set of subscriptions considered to be in the group, and are also the attributes that cannot be altered if a subscription is grouped. Alteration of SubLevel results in MQRC_SUBLEVEL_NOT_ALTERABLE, and alteration of any of the others (which can be changed if a subscription is not grouped) results in MQRC_GROUPING_NOT_ALTERABLE.
5. Fields in the MQSD are filled in on return from an MQSUB call which uses the MQSO_RESUME option. The MQSD returned can be passed directly into an MQSUB call which uses the MQSO_ALTER option with any changes you need to make to the subscription applied to the MQSD. Some fields have special considerations as noted in the table.

MQSD output from MQSUB

Field name in MQSD	Special considerations
Access or creation options	None of these options are set on return from the MQSUB call. If you subsequently reuse the MQSD in an MQSUB call the option you require must be explicitly set.
Durability options, Destination options, Registration Options & Wildcard options	These options will be set as appropriate
Publication options	These options will be set as appropriate, with the exception of MQSO_NEW_PUBLICATIONS_ONLY which is only applicable to MQSO_CREATE.
Other options	These options are unchanged on return from an MQSUB call. They control how the API call is issued and are not stored with the subscription. They must be set as required on any subsequent MQSUB call reusing the MQSD.
ObjectName	This input only field is unchanged on return from an MQSUB call.
ObjectString	This input only field is unchanged on return from an MQSUB call. The Full topic name used is returned in the <i>ResObjectString</i> field, if a buffer is provided.
AlternateUserId and AlternateSecurityId	These input only fields are unchanged on return from an MQSUB call. They control how the API call is issued and are not stored with the subscription. They must set as required on any subsequent MQSUB call reusing the MQSD.
SubExpiry	On return from an MQSUB call using the MQSO_RESUME option this field will be set to the original expiry of the subscription and not the remaining expiry time. If you subsequently reuse the MQSD in an MQSUB call using the MQSO_ALTER option you will reset the expiry of the subscription to start counting down again.
SubName	This field is an input field on an MQSUB call and is not changed on output.
SubUserData and SelectionString	<p>These variable length fields will be returned on output from an MQSUB call using the MQSO_RESUME option, if a buffer is provided, and also a positive buffer length in <i>VSBufSize</i>. If no buffer is provided only the length will be returned in the <i>VSLength</i> field of the MQCHARV. If the buffer provided is smaller than the space required to return the field, only <i>VSBufSize</i> bytes are returned in the provided buffer.</p> <p>If you subsequently reuse the MQSD in an MQSUB call using the MQSO_ALTER option and a buffer is not provided but a non-zero <i>VSLength</i> is provided, if that length matches the existing length of the field, no alteration will made to the field.</p>

| MQSD output from MQSUB

Field name in MQSD	Special considerations
SubCorrelId and PubAccountingToken	If you do not use MQSO_SET_CORREL_ID, then the <i>SubCorrelId</i> will be generated by the queue manager. If you do not use MQSO_SET_IDENTITY_CONTEXT, then the <i>PubAccountingToken</i> will be generated by the queue manager. These fields will be returned in the MQSD from an MQSUB call using the MQSO_RESUME option. If they are generated by the queue manager, the generated value will be returned on an MQSUB call using the MQSO_CREATE or MQSO_ALTER option.
PubPriority, SubLevel & PubAppIdentityData	These fields will be returned in the MQSD.
ResObjectString	This output only field will be returned in the MQSD if a buffer is provided.

Language invocations for MQSUB

C invocation

```
MQSUB (Hconn, &SubDesc, &Hobj, &Hsub, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHCONN Hconn; /* Connection handle */
MQSD SubDesc; /* Subscription descriptor */
MQHOBJ Hobj; /* Object handle */
MQHOBJ Hsub; /* Subscription handle */
MQLONG CompCode; /* Completion code */
MQLONG Reason; /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQSUB' USING HCONN, SUBDESC, HOBJ, HSUB, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN PIC S9(9) BINARY.
** Subscription descriptor
01 SUBDESC.
COPY CMQSDV.
** Object handle
01 HOBJ PIC S9(9) BINARY.
** Subscription handle
01 HSUB PIC S9(9) BINARY.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON PIC S9(9) BINARY.
```

PL/I invocation

```
call MQSUB (Hconn, SubDesc, Hobj, Hsub, CompCode, Reason)
```

Declare the parameters as follows:

```
dc1 Hconn fixed bin(31); /* Connection handle */
dc1 SubDesc like MQSD; /* Subscription descriptor */
dc1 Hobj fixed bin(31); /* Object handle */
dc1 Hsub fixed bin(31); /* Subscription handle */
dc1 CompCode fixed bin(31); /* Completion code */
dc1 Reason fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 assembler invocation

CALL MQSUB, (HCONN, SUBDESC, HOBJ, HSUB, COMPCODE, REASON)

Declare the parameters as follows:

HCONN	DS	F	Connection handle
SUBDESC	CMQSDA	,	Subscription descriptor
HOBJ	DS	F	Object handle
HSUB	DS	F	Subscription handle
COMPCODE	DS	F	Completion code
REASON	DS	F	Reason code qualifying COMPCODE

MQSUBRQ - Subscription request

The MQSUBRQ call makes a request for a subscription, when the subscriber has been registered with MQSO_PUBLICATIONS_ON_REQUEST.

Syntax for MQSUBRQ

MQSUBRQ (*Hconn, Hsub, Action, SubRqOpts, CompCode, Reason*)

Parameters for MQSUBRQ

The MQSUBRQ call has the following parameters.

Hconn (MQHCONN) - Input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on i5/OS for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Hsub (MQHOBJ) - input

This handle represents the subscription for which an update is to be requested. The value of *Hsub* was returned from a previous MQSUB call.

Action (MQLONG) - Input

This parameter controls the particular action that is being requested on the subscription. The following value must be specified:

MQSR_ACTION_PUBLICATION

This action requests an update publication be sent for the specified topic. It can only be used if the subscriber specified the option MQSO_PUBLICATIONS_ON_REQUEST on the MQSUB call when it made the subscription. If the queue manager has a retained publication for the topic, this is sent to the subscriber. If not, the call fails. If an application is sent a publication which was retained, this will be indicated by the MQIsRetained message property of that publication.

Since the topic in the existing subscription represented by the Hsub parameter may contain wildcards, the subscriber might receive multiple retained publications.

SubRqOpts (MQSRO) - Input/output

These options control the action of MQSUBRQ, see “MQSRO - Subscription request options” on page 359 for details.

If no options are required, programs written in C or S/390 assembler can specify a null parameter address instead of specifying the address of an MQSRO structure.

CompCode (MQLONG) - output

The completion code; it is one of the following:

MQCC_OK

Successful completion

MQCC_WARNING

Warning (partial completion)

MQCC_FAILED

Call failed

Reason (MQLONG) - output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_FUNCTION_NOT_SUPPORTED

2298 (X'08FA') The function requested is not available in the current environment.

MQRC_NO_RETAINED_MSG

2437 (X'0985') There are no retained publications currently stored for this topic.

MQRC_OPTIONS_ERROR

2046 (X'07FE') Options parameter or field contains options that are not valid, or a combination of options that is not valid.

MQRC_Q_MGR QUIESCING

2161 (X'0871') Queue manager quiescing

MQRC_SRO_ERROR

2438 (X'0986') On the MQSUBRQ call, the Subscription Request Options MQSRO is not valid.

Usage notes for MQSUBRQ

The following usage notes apply to the use of the Action code MQSR_ACTION_PUBLICATION:

1. If this verb completes successfully, the retained publications matching the subscription specified have been sent to the subscription and can be received by using MQGET or MQCB using the Hobj returned on the original MQSUB verb that created the subscription.
2. If the topic subscribed to by the original MQSUB verb that created the subscription contained a wildcard, more than one retained publication may be sent. The number of publications sent as a result of this call is recorded in the NumPubs field in the SubRqOpts structure.
3. If this verb completes with a reason code of MQRC_NO_RETAINED_MSG then there were no currently retained publications for the topic specified.#
4. If this verb completes with a reason code of MQRC_RETAINED_MSG_Q_ERROR or MQRC_RETAINED_NOT_DELIVERED then there are currently retained publications for the topic specified but an error has occurred that that meant they were unable to be delivered.
5. The application must have a current subscription to the topic before it can make this call. If the subscription was made in a previous instance of the application and a valid handle to the subscription is not available, the application must first call MQSUB with the MQSO_RESUME option to obtain a handle to it for use in this call.
6. The publications are sent to the destination that is registered for use with the current subscription of this application. If the publications should be sent somewhere else, the subscription must first be altered using the MQSUB call with the MQSO_ALTER option.

Language invocations for MQSUBRQ

C invocation

```
MQSUB (Hconn, Hsub, Action, &SubRqOpts, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHCONN Hconn; /* Connection handle */
MQHOBJ Hsub; /* Subscription handle */
MQLONG Action; /* Action requested by MQSUBRQ */
MQSRO SubRqOpts; /* Options that control the action of MQSUBRQ */
MQLONG CompCode; /* Completion code */
MQLONG Reason; /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQSUBRQ' USING HCONN, HSUB, ACTION, SUBRQOPTS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN PIC S9(9) BINARY.
** Subscription handle
01 HSUB PIC S9(9) BINARY.
** Action requested by MQSUBRQ
01 ACTION PIC S9(9) BINARY.
** Options that control the action of MQSUBRQ
01 SUBRQOPTS.
COPY CMQSROV.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON PIC S9(9) BINARY.
```

PL/I invocation

```
call MQSUBRQ (Hconn, Hsub, Action, SubRqOpts, CompCode, Reason)
```

Declare the parameters as follows:

```
dc1 Hconn fixed bin(31); /* Connection handle */
dc1 Hsub fixed bin(31); /* Subscription handle */
dc1 Action fixed bin(31); /* Action requested by MQSUBRQ */
dc1 SubRqOpts like MQSRO; /* Options that control the action of MQSUBRQ */
dc1 CompCode fixed bin(31); /* Completion code */
dc1 Reason fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
CALL MQSUBRQ,(HCONN, HSUB, ACTION, SUBRQOPTS,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN DS F Connection handle
HSUB DS F Subscription handle
ACTION DS F Action requested by MQSUBRQ
SUBRQOPTS CMQSROA , Options that control the action of MQSUBRQ
COMP CODE DS F Completion code
REASON DS F Reason code qualifying COMP CODE
```

Chapter 3. Attributes of objects

This collection of topics lists only those WebSphere MQ objects that can be the subject of an MQINQ function call, and gives details of the attributes that can be inquired on and the selectors to be used.

Attributes for the queue manager

Some queue-manager attributes are fixed for particular implementations; others can be changed by using the MQSC command ALTER QMGR.

The attributes can also be displayed by using the command DISPLAY QMGR. Most queue-manager attributes can be inquired by opening a special MQOT_Q_MGR object, and using the MQINQ call with the handle returned.

The following table summarizes the attributes that are specific to the queue manager. The attributes are described in alphabetic order.

Note: The names of the attributes shown in this book are descriptive names used with the MQINQ call; the names are the same as for the PCF commands. When MQSC commands are used to define, alter, or display attributes, alternative short names are used; see the *WebSphere MQ Script (MQSC) Command Reference* for details.

Table 95. Attributes for the queue manager

Attribute	Description	Topic
<i>AccountingConnOverride</i>	Override accounting settings.	AccountingConnOverride
<i>AccountingInterval</i>	How often to write intermediate accounting records.	AccountingInterval
<i>AdoptNewMCACheck</i>	Elements checked to determine whether to adopt new MCA.	AdoptNewMCACheck
<i>AdoptNewMCAType</i>	Whether to restart automatically an orphaned instance of an MCA of a given channel type.	AdoptNewMCAType
<i>AlterationDate</i>	Date when definition was last changed	AlterationDate
<i>AlterationTime</i>	Time when definition was last changed	AlterationTime
<i>AuthorityEvent</i>	Controls whether authorization (Not Authorized) events are generated	AuthorityEvent
<i>BridgeEvent</i>	Control attribute for bridge events.	BridgeEvent
<i>ChannelAutoDef</i>	Controls whether automatic channel definition is permitted	ChannelAutoDef
<i>ChannelAutoDefEvent</i>	Controls whether channel automatic-definition events are generated	ChannelAutoDefEvent
<i>ChannelAutoDefExit</i>	Name of user exit for automatic channel definition	ChannelAutoDefExit
<i>ChannelEvent</i>	Control attribute for channel events.	ChannelEvent
<i>ChannelInitiatorControl</i>	Control attribute for channel initiator	ChannelInitiatorControl
<i>ChannelMonitoring</i>	Online monitoring data for channels	ChannelMonitoring
<i>ChannelStatistics</i>	Controls collection of statistics data for channels.	ChannelStatistics
<i>ChinitAdapters</i>	Number of adapter subtasks for processing WebSphere MQ calls.	ChinitAdapters
<i>ChinitDispatchers</i>	Number of dispatchers to use for the channel initiator.	ChinitDispatchers
<i>ChinitServiceParm</i>	Reserved for IBM use.	

Table 95. Attributes for the queue manager (continued)

Attribute	Description	Topic
<i>ChinitTraceAutoStart</i>	Whether channel initiator trace should start automatically.	ChinitTraceAutoStart
<i>ChinitTraceTableSize</i>	Size of channel initiator's trace data space.	ChinitTraceTableSize
<i>ClusterSenderMonitoringDefault</i>	Online monitoring data default for cluster sender channels	ClusterSenderMonitoringDefault
<i>ClusterSenderStatistics</i>	Controls collection of statistics monitoring information for cluster sender channels.	ClusterSenderStatistics
<i>ClusterWorkloadData</i>	User data for cluster workload exit	ClusterWorkloadData
<i>ClusterWorkloadExit</i>	Name of user exit for cluster workload management	ClusterWorkloadExit
<i>ClusterWorkloadLength</i>	Maximum length of message data passed to cluster workload exit	ClusterWorkloadLength
<i>CLWLMRUChannels</i>	Number of most recently used channels for cluster workload balancing	CLWLMRUChannels
<i>CLWLUseQ</i>	Cluster workload use remote queue.	CLWLUseQ
<i>CodedCharSetId</i>	Coded character set identifier	CodedCharSetId
<i>CommandEvent</i>	Control attribute for command events.	CommandEvent
<i>CommandInputQName</i>	Command input queue name	CommandInputQName attribute
<i>CommandLevel</i>	Command level	CommandLevel
<i>CommandServerControl</i>	Control attribute for command server.	CommandServerControl attribute
<i>ConfigurationEvent</i>	Control attribute for configuration events.	Configuration Event attribute
<i>DeadLetterQName</i>	Name of dead-letter queue	DeadLetterQName
<i>DefXmitQName</i>	Default transmission queue name	DefXmitQName
<i>DistLists</i>	Distribution list support	DistLists
<i>DNSGroup</i>	Name of group for TCP listener when using Workload Manager Dynamic Domain Name Services support.	DNSGroup
<i>DNSWLM</i>	Whether TCP listener registers with Workload Manager for Dynamic Domain Name Services.	DNSWLM
<i>ExpiryInterval</i>	Interval between scans for expired messages	ExpiryInterval
<i>IGQPutAuthority</i>	Intra-group queuing put authority	IGQPutAuthority
<i>IGQUserId</i>	Intra-group queuing user identifier	IGQUserId
<i>InhibitEvent</i>	Controls whether inhibit (Inhibit Get and Inhibit Put) events are generated	InhibitEvent
<i>IPAddressVersion</i>	Version of the internet protocol address	IPAddressVersion
<i>IntraGroupQueuing</i>	Intra-group queuing support	IntraGroupQueuing
<i>ListenerTimer</i>	Time interval between attempts to restart listener after APPC or TCP/IP failure.	ListenerTimer
<i>LocalEvent</i>	Controls whether local error events are generated	LocalEvent
<i>LoggerEvent</i>	Controls whether logger events are generated	LoggerEvent
<i>LUGroupName</i>	Generic LU name for LU 6.2 listener that handles inbound transmissions for queue-sharing group.	LUGroupName
<i>LUName</i>	Name of LU to use for outbound LU 6.2 transmissions.	LUName
<i>LU62ARMSuffix</i>	Suffix of SYS1.PARMLIB member APPCPMxx, that nominates LUADD for this channel initiator.	LU62ARMSuffix
<i>LU62Channels</i>	Maximum number of current channels or connected clients that use LU 6.2.	LU62Channels
<i>MaxActiveChannels</i>	Maximum number of channels that can be active at any time.	MaxActiveChannels
<i>MaxChannels</i>	Maximum number of current channels.	MaxChannels
<i>MaxHandles</i>	Maximum number of handles	MaxHandles
<i>MaxMsgLength</i>	Maximum message length in bytes	MaxMsgLength

Table 95. Attributes for the queue manager (continued)

Attribute	Description	Topic
<i>MaxPriority</i>	Maximum priority	MaxPriority attribute
<i>MaxPropertiesLength</i>	Maximum length of property data in bytes	
<i>MaxUncommittedMsgs</i>	Maximum number of uncommitted messages within a unit of work	MaxUncommittedMsgs
<i>MQIAccounting</i>	Controls collection of accounting information for MQI data.	MQIAccounting
<i>MQIStatistics</i>	Controls collection of statistics monitoring information for queue manager.	MQIStatistics
<i>MsgMarkBrowseInterval</i>	Interval after which the queue manager can remove the mark from browsed messages.	MsgMarkBrowseInterval
<i>OutboundPortMax</i>	With <i>OutboundPortMin</i> , defines range of port numbers to use when binding outgoing channels.	OutboundPortMin
<i>OutboundPortMin</i>	With <i>OutboundPortMax</i> , defines range of port numbers to use when binding outgoing channels.	OutboundPortMin
<i>PerformanceEvent</i>	Controls whether performance-related events are generated	PerformanceEvent
<i>Platform</i>	Platform on which the queue manager is running	Platform
<i>PubSubNPInputMsg</i>	Whether to discard (or keep) an undelivered input message	PubSubNPInputMsg
<i>PubSubNPResponse</i>	Controls the behavior of undelivered	PubSubNPResponse
<i>PubSubMaxMsgRetryCount</i>	The number of retries when processing (under syncpoint) a failed command message	PubSubMaxMsgRetryCount
<i>PubSubSyncPoint</i>	Whether only persistent (or all) messages should be processed under syncpoint	PubSubSyncPoint
<i>PubSubMode</i>	Whether the queued publish/subscribe interface is running	PubSubMode
<i>QMGrDesc</i>	Queue manager description	QMGrDesc
<i>QMGrIdentifier</i>	Unique internally-generated identifier of queue manager	QMGrIdentifier
<i>QMGrName</i>	Queue manager name	QMGrName
<i>QSGName</i>	Name of queue-sharing group	QSGName
<i>QueueAccounting</i>	Controls collection of accounting information for queues.	QueueAccounting
<i>QueueMonitoring</i>	Online monitoring data for queues	QueueMonitoring
<i>QueueStatistics</i>	Controls collection of statistics data for queues.	QueueStatistics
<i>ReceiveTimeout</i>	How long TCP/IP channel waits for data before returning to inactive state.	ReceiveTimeout
<i>ReceiveTimeoutMin</i>	Qualifier for <i>ReceiveTimeout</i> .	ReceiveTimeoutMin
<i>ReceiveTimeoutType</i>	Minimum time that TCP/IP channel waits for data before returning to inactive state.	ReceiveTimeoutType
<i>RemoteEvent</i>	Controls whether remote error events are generated	RemoteEvent
<i>RepositoryName</i>	Name of cluster for which this queue manager provides repository services	RepositoryName
<i>RepositoryNameList</i>	Name of namelist object containing names of clusters for which this queue manager provides repository services	RepositoryNameList
<i>ScyCase</i>	Case of security profiles	ScyCase
<i>SharedQMGrName</i>	Shared queue queue-manager name	SharedQMGrName
<i>SSLCRLNameList</i>	Name of namelist object containing names of authentication information objects.	Note 1
<i>SSLCryptoHardware</i>	Cryptographic hardware configuration string.	Note 1
<i>SSLEvent</i>	Control attribute for SSL events.	SSLEvent
<i>SSLFIPSRequired</i>	Use only FIPS-certified algorithms for cryptography.	SSLFIPSRequired

Table 95. Attributes for the queue manager (continued)

Attribute	Description	Topic
<i>SSLKeyRepository</i>	Location of SSL key repository.	Note 1
<i>SSLKeyResetCount</i>	SSL key reset count.	SSLKeyResetCount
<i>SSLTasks</i>	Number of server subtasks for processing SSL calls.	Note 1
<i>StatisticsInterval</i>	How often to write statistics monitoring data.	StatisticsInterval
<i>StartStopEvent</i>	Controls whether start and stop events are generated	StartStopEvent
<i>SyncPoint</i>	Syncpoint availability	SyncPoint
<i>TCPChannels</i>	Maximum number of current channels or connected clients that use TCP/IP.	TCPChannels
<i>TCPKeepAlive</i>	Whether to use TCP KEEPALIVE to check other end of connection.	TCPKeepAlive
<i>TCPName</i>	Name of TCP/IP system that you are using.	TCPName
<i>TCPStackType</i>	How channel initiator can use TCP/IP addresses.	TCPStackType
<i>TraceRouteRecording</i>	Controls recording of trace-route information.	TraceRouteRecording attribute
<i>TriggerInterval</i>	Trigger-message interval	TriggerInterval
Notes:		
1. This attribute cannot be inquired using the MQINQ call, and is not described in this book. See WebSphere MQ Programmable Command Formats and Administration Interface for details of this attribute.		

AccountingConnOverride (MQLONG)

This allows applications to override the setting of the ACCTMQI and ACCTQDATA values in the Qmgr attribute.

The value is one of the following:

MQMON_DISABLED

Applications cannot override the setting of the ACCTMQI and ACCTQ Qmgr attributes using the Options field in the MQCNO structure on the MQCONN call. This is the default value.

MQMON_ENABLED

Applications can override the ACCTQ and ACCTMQI Qmgr attributes using the Options field in the MQCNO structure.

Changes to this value are only effective for connections to the queue manager after the change to the attribute.

This attribute is supported only on i5/OS, Unix systems, and Windows.

To determine the value of this attribute, use the MQIA_ACCOUNTING_CONN_OVERRIDE selector with the MQINQ call.

AccountingInterval (MQLONG)

This specifies how long before intermediate accounting records are written (in seconds).

The value is an integer in the range 0 to 604800, with a default value of 1800 (30 minutes). Specify 0 to turn off intermediate records.

This attribute is supported only on i5/OS, Unix systems, and Windows.

To determine the value of this attribute, use the MQIA_ACCOUNTING_INTERVAL selector with the MQINQ call.

AdoptNewMCACheck (MQLONG)

This defines the elements to check to determine whether to adopt an MCA when a new inbound channel is detected that has the same name as an MCA that is already active

The value is one of the following:

MQADOPT_CHECK_Q_MGR_NAME
Check the queue manager name.

MQADOPT_CHECK_NET_ADDR
Check the network address.

MQADOPT_CHECK_ALL
Check the queue manager name and network address. If possible, perform this check to protect your channels from being shut down, inadvertently or maliciously. This is the default value.

MQADOPT_CHECK_NONE
Do not check any elements.

Changes to this attribute take effect the next time that a channel attempts to adopt a channel.

This attribute is supported only on z/OS.

To determine the value of this attribute, use the MQIA_ADOPTNEWMCA_CHECK selector with the MQINQ call.

AdoptNewMCAType (MQLONG)

This specifies whether to restart automatically an orphaned instance of an MCA of a given channel type when a new inbound channel request matching the AdoptNewMCACheck attribute is detected

The value is one of the following:

MQADOPT_TYPE_NO
Adopting orphaned channel instances is not required. This is the default value.

MQADOPT_TYPE_ALL
Adopt all channel types.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_ADOPTNEWMCA_TYPE selector with the MQINQ call.

AlterationDate (MQCHAR12)

This is the date when the definition was last changed. The format of the date is YYYY-MM-DD, padded with two trailing blanks to make the length 12 bytes.

To determine the value of this attribute, use the MQCA_ALTERATION_DATE selector with the MQINQ call. The length of this attribute is given by MQ_DATE_LENGTH.

AlterationTime (MQCHAR8)

This is the time when the definition was last changed. The format of the time is HH.MM.SS.

To determine the value of this attribute, use the MQCA_ALTERATION_TIME selector with the MQINQ call. The length of this attribute is given by MQ_TIME_LENGTH.

AuthorityEvent (MQLONG)

This controls whether authorization (Not Authorized) events are generated. The value is one of the following:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see *WebSphere MQ Monitoring*.

To determine the value of this attribute, use the MQIA_AUTHORITY_EVENT selector with the MQINQ call.

BridgeEvent (MQLONG)

This specifies whether IMS bridge events are generated.

The value is one of the following:

MQEVR_ENABLED
Generate IMS bridge events, as follows:
MQRC_BRIDGE_STARTED
MQRC_BRIDGE_STOPPED

MQEVR_DISABLED
Do not generate IMS bridge events; this is the default value.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_BRIDGE_EVENT selector with the MQINQ call.

ChannelAutoDef (MQLONG)

This attribute controls the automatic definition of channels of type MQCHT_RECEIVER and MQCHT_SVRCONN. Automatic definition of MQCHT_CLUSSDR channels is always enabled. The value is one of the following:

MQCHAD_DISABLED
Channel auto-definition disabled.

MQCHAD_ENABLED

Channel auto-definition enabled.

This attribute is supported only on AIX, HP-UX, i5/OS, Linux, Solaris, and Windows.

To determine the value of this attribute, use the MQIA_CHANNEL_AUTO_DEF selector with the MQINQ call.

ChannelAutoDefEvent (MQLONG)

This controls whether channel automatic-definition events are generated. It applies to channels of type MQCHT_RECEIVER, MQCHT_SVRCONN, and MQCHT_CLUSSDR. The value is one of the following:

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

For more information about events, see *WebSphere MQ Monitoring*.

This attribute is supported only on AIX, HP-UX, i5/OS, Linux, Solaris, and Windows.

To determine the value of this attribute, use the MQIA_CHANNEL_AUTO_DEF_EVENT selector with the MQINQ call.

ChannelAutoDefExit (MQCHARn)

This is the name of the user exit for automatic channel definition. If this name is nonblank, and *ChannelAutoDef* has the value MQCHAD_ENABLED, the exit is called each time that the queue manager is about to create a channel definition. This applies to channels of type MQCHT_RECEIVER, MQCHT_SVRCONN, and MQCHT_CLUSSDR. The exit can then do one of the following:

- Create the channel definition 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 *WebSphere MQ Intercommunications* for details of the value of this attribute in various environments.

This attribute is supported only on AIX, HP-UX, i5/OS, Linux, Solaris, Windows, and z/OS. On z/OS, it applies only to cluster-sender and cluster-receiver channels.

To determine the value of this attribute, use the MQCA_CHANNEL_AUTO_DEF_EXIT selector with the MQINQ call. The length of this attribute is given by MQ_EXIT_NAME_LENGTH.

ChannelEvent (MQLONG)

This specifies whether channel events are generated.

The value is one of the following:

MQEVR_EXCEPTION

Only generate the following channel events:

- MQRC_CHANNEL_ACTIVATED
 - MQRC_CHANNEL_CONV_ERROR
 - MQRC_CHANNEL_NOT_ACTIVATED
 - MQRC_CHANNEL_STOPPED with the following ReasonQualifiers:
 - MQRQ_CHANNEL_STOPPED_ERROR
 - MQRQ_CHANNEL_STOPPED_RETRY
 - MQRQ_CHANNEL_STOPPED_DISABLED
- MQRC_CHANNEL_STOPPED_BY_USER

MQEVR_ENABLED

Generate all channel events. That is, in addition to those generated by EXCEPTION, generate the following channel events:

- MQRC_CHANNEL_STARTED
- MQRC_CHANNEL_STOPPED with the following ReasonQualifier:
 - MQRQ_CHANNEL_STOPPED_OK

MQEVR_DISABLED

Do not generate channel events; this is the default value.

To determine the value of this attribute, use the MQIA_CHANNEL_EVENT selector with the MQINQ call.

ChannelInitiatorControl (MQLONG)

This specifies whether the channel initiator is to be started when the queue manager starts.

The value is one of the following:

MQSVC_CONTROL_MANUAL

The channel initiator is not to be started automatically.

MQSVC_CONTROL_Q_MGR

The channel initiator is to be started automatically when the queue manager starts.

To determine the value of this attribute, use the MQIA_CHINIT_CONTROL selector with the MQINQ call.

ChannelMonitoring (MQLONG)

This specifies online monitoring data for channels.

The value is one of the following:

MQMON_NONE

Disable data collection for channel monitoring for all channels regardless of the setting of the STATCHL channel attribute. This is the default value.

MQMON_OFF

Turn monitoring data collection off for channels that specify QMGR in the STATCHL channel attribute.

MQMON_LOW

Turn monitoring data collection on with a low ratio of data collection for channels specifying QMGR in the STATCHL channel attribute.

MQMON_MEDIUM

Turn monitoring data collection on with a moderate ratio of data collection for channels specifying QMGR in the STATCHL channel attribute.

MQMON_HIGH

Turn monitoring data collection on with a high ratio of data collection for channels specifying QMGR in the STATCHL channel attribute.

To determine the value of this attribute, use the MQIA_MONITORING_CHANNEL selector with the MQINQ call.

ChannelStatistics (MQLONG)

This controls the collection of statistics data for channels.

The value is one of the following:

MQMON_NONE

Disable data collection for channel statistics for all channels regardless of the setting of the STATCHL channel attribute. This is the default value.

MQMON_OFF

Turn statistics data collection off for channels that specify QMGR in the STATCHL channel attribute.

MQMON_LOW

Turn statistics data collection on with a low ratio of data collection for channels specifying QMGR in the STATCHL channel attribute.

MQMON_MEDIUM

Turn statistics data collection on with a moderate ratio of data collection for channels specifying QMGR in the STATCHL channel attribute.

MQMON_HIGH

Turn statistics data collection on with a high ratio of data collection for channels specifying QMGR in the STATCHL channel attribute.

For most systems you are recommended to use MEDIUM. However, for a channel that processes a high volume of messages each second, you might want to reduce the sampling level by selecting LOW. Also, for a channel that processes only a few messages, and for which the most current information is important, you might want to select HIGH.

This attribute is supported only on i5/OS, UNIX systems, and Windows.

To determine the value of this attribute, use the MQIA_STATISTICS_CHANNEL selector with the MQINQ call.

ChinitAdapters (MQLONG)

This is the number of adapter subtasks to use to process WebSphere MQ calls. The value must be between 0 and 9999, with a default value of 8.

The ratio of adapters to dispatchers (the ChinitDispatchers attribute) should be about 8 to 5. However, if you have only a small number of channels, you do not have to decrease the value of this parameter from the default value. You are recommended to use the following values: for a test system, 8 (default); for a production system, 20. Ideally, you should have 20 adapters, which gives greater

parallelism of WebSphere MQ calls. This is particularly important for persistent messages. Fewer adapters might be better for nonpersistent messages.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_CHINIT_ADAPTERS selector with the MQINQ call.

ChinitDispatchers (MQLONG)

This is the number of dispatchers to use for the channel initiator. The value must be between 0 and 9999, with a default value of 5.

As a guideline, allow one dispatcher for 50 current channels. However, if you have only a small number of channels, you do not have to decrease the value of this attribute from the default value. If you are using TCP/IP, the greatest number of dispatchers that are used for TCP/IP channels is 100, even if you specify a larger value here. You are recommended to use the following settings: test systems, 5 (the default); production systems, 20 (you need 20 dispatchers to handle up to 1000 active channels).

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_CHINIT_DISPATCHERS selector with the MQINQ call.

ChinitTraceAutoStart (MQLONG)

This specifies whether to start channel initiator trace automatically.

The value is one of the following:

MQTRAXSTR_YES

Start channel initiator trace automatically. This is the default value.

MQTRAXSTR_NO

Do not start channel initiator trace automatically.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_CHINIT_TRACE_AUTO_START selector with the MQINQ call.

ChinitTraceTableSize (MQLONG)

This is the size of the channel initiator's trace data space (in MB). The value must be between zero and 2048, with a default value of 2.

Note: Whenever you use large z/OS data spaces, ensure that you have sufficient auxiliary storage on your system to support any related z/OS paging activity. You might also need to increase the size of your SYS1.DUMP data sets.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_CHINIT_TRACE_TABLE_SIZE selector with the MQINQ call.

ClusterSenderMonitoringDefault (MQLONG)

This specifies the value to be substituted for the *ChannelMonitoring* attribute of automatically-defined cluster sender channels.

The value is one of the following:

MQMON_Q_MGR

Collection of online monitoring data is inherited from the setting of the queue manager *ChannelMonitoring* attribute. This is the default value.

MQMON_OFF

Monitoring for the channel is switched off

MQMON_LOW

Unless *ChannelMonitoring* is *MQMON_NONE*, monitoring is switched on with a low rate of data collection with a minimal impact on system performance. The data collected is not likely to be the most current.

MQMON_MEDIUM

Unless *ChannelMonitoring* is *MQMON_NONE*, monitoring is switched on with a moderate rate of data collection with limited impact on system performance.

MQMON_HIGH

Unless *ChannelMonitoring* is *MQMON_NONE*, monitoring is switched on with a high rate of data collection with a likely impact on system performance. The data collected is the most current available.

To determine the value of this attribute, use the *MQIA_MONITORING_AUTO_CLUSSDR* selector with the *MQINQ* call.

ClusterSenderStatistics (MQLONG)

Because cluster sender channels can be automatically defined from the definition of *CLUSRCVR* in the repository, you cannot alter the setting of the *STATCHL* attribute for these auto-defined cluster sender channels using *ALTER channel*. For these channels the decision of whether to collect online monitoring data is based on the setting of this queue manager attribute.

The value is one of the following:

MQMON_Q_MGR

Statistics data collection for auto-defined cluster sender channels is based on the value of the queue manager attribute *STATCHL*. This is the default value.

MQMON_OFF

Switch off statistics data collection for auto-defined cluster sender channels.

MQMON_LOW

Switch on statistics data collection for auto-defined cluster sender channels with a low ratio of data collection.

MQMON_MEDIUM

Switch on statistics data collection for auto-defined cluster sender channels with a moderate ratio of data collection.

MQMON_HIGH

Switch on statistics data collection for auto-defined cluster sender channels with a high ratio of data collection.

For most systems we recommend MEDIUM. However, for an auto-defined cluster sender channel that processes a high volume of messages each second, you might want to reduce the sampling level by selecting LOW. Also, for a channel that processes only a few messages, and for which the most current information is important, you might want to select HIGH.

To determine the value of this attribute, use the MQIA_STATISTICS_AUTO_CLUSSDR selector with the MQINQ call.

ClusterWorkloadData (MQCHAR32)

This is a user-defined 32-byte character string that is passed to the cluster workload exit when it is called. If there is no data to pass to the exit, the string is blank.

This attribute is supported only on AIX, HP-UX, i5/OS, Linux, Solaris, Windows and z/OS.

To determine the value of this attribute, use the MQCA_CLUSTER_WORKLOAD_DATA selector with the MQINQ call.

ClusterWorkloadExit (MQCHARn)

This is the name of the user exit for cluster workload management. If this name is nonblank, the exit is called each time that a message is put to a cluster queue or moved from one cluster-sender queue to another. The exit can then decide whether to accept the queue instance selected by the queue manager as the destination for the message, or choose another queue instance.

Note: Both the length and the value of this attribute are environment specific. See *WebSphere MQ Intercommunications* for details of the value of this attribute in various environments.

This attribute is supported only on AIX, HP-UX, i5/OS, Linux, Solaris, Windows and z/OS.

To determine the value of this attribute, use the MQCA_CLUSTER_WORKLOAD_EXIT selector with the MQINQ call. The length of this attribute is given by MQ_EXIT_NAME_LENGTH.

ClusterWorkloadLength (MQLONG)

This is the maximum length of message data that is passed to the cluster workload exit. The actual length of data passed to the exit is the minimum of the following:

- The length of the message.
- The queue-manager's *MaxMsgLength* attribute.
- The *ClusterWorkloadLength* attribute.

This attribute is supported only on AIX, HP-UX, i5/OS, Linux, Solaris, Windows and z/OS.

To determine the value of this attribute, use the MQIA_CLUSTER_WORKLOAD_LENGTH selector with the MQINQ call.

CLWLMRUChannels (MQLONG)

This specifies the maximum number of most-recently-used cluster channels, to be considered for use by the cluster workload choice algorithm. This is a value between 1 and 999999999. For more information on using this attribute, see *WebSphere MQ Queue Manager Clusters*.

To determine the value of this attribute, use the MQIA_CLWL_MRU_CHANNELS selector with the MQINQ call.

CLWLUseQ (MQLONG)

This specifies whether to use remote queues for the cluster workload.

The value is one of the following:

MQCLWL_USEQ_ANY

Use both local and remote queues.

MQCLWL_USEQ_LOCAL

Do not use remote queues. This is the default value.

To determine the value of this attribute, use the MQIA_CLWL_USEQ selector with the MQINQ call.

CodedCharSetId (MQLONG)

This defines the character set used by the queue manager for all character string fields defined in the MQI, including the names of objects, queue creation date and time, and so on. The character set must be one that has single-byte characters for the characters that are valid in object names. It does not apply to application data carried in the message. The value depends on the environment:

- On z/OS, the value is set from the system parameters when the queue manager is started; the default value is 500. Refer to the *WebSphere MQ for z/OS System Setup Guide* for further information.
- On Windows, the value is the primary CODEPAGE of the user creating the queue manager.
- On i5/OS, the value is that which is set in the environment when the queue manager is first created.
- On UNIX systems, the value is the default CODESET for the locale of the user creating the queue manager.

To determine the value of this attribute, use the MQIA_CODED_CHAR_SET_ID selector with the MQINQ call.

CommandEvent (MQLONG)

This specifies whether command events are generated, as follows:

MQEVR_DISABLED

Do not generate command events. This is the default.

MQEVR_ENABLED

Generate command events.

MQEVR_NO_DISPLAY

Command events are generated for all successful commands other than MQINQ.

This attribute is supported only on z/OS.

To determine the value of this attribute, use the MQIA_COMMAND_EVENT selector with the MQINQ call.

CommandInputQName (MQCHAR48)

This is the name of the command input queue defined on the local queue manager. This is a queue to which users can send commands, if authorized to do so. The name of the queue depends on the environment:

- On z/OS, the name of the queue is SYSTEM.COMMAND.INPUT; MQSC and PCF commands can be sent to it. Refer to *WebSphere MQ Script (MQSC) Command Reference* for details of MQSC commands and *WebSphere MQ Programmable Command Formats and Administration Interface* for details of PCF commands.
- In all other environments, the name of the queue is SYSTEM.ADMIN.COMMAND.QUEUE, and only PCF commands can be sent to it. However, an MQSC command can be sent to this queue if the MQSC command is enclosed within a PCF command of type MQCMD_ESCAPE. Refer to *WebSphere MQ Programmable Command Formats and Administration Interface* for details of the Escape command.

To determine the value of this attribute, use the MQCA_COMMAND_INPUT_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

CommandLevel (MQLONG)

This indicates the level of system control commands supported by the queue manager. The value is one of the following:

MQCMDL_LEVEL_1

Level 1 of system control commands.

This value is returned by the following:

- MQSeries for AIX Version 2 Release 2
- MQSeries for
 - Version 1 Release 1.1
 - Version 1 Release 1.2
 - Version 1 Release 1.3
- MQSeries for OS/2 Version 2 Release 0
- MQSeries for OS/400®
 - Version 2 Release 3
 - Version 3 Release 1
 - Version 3 Release 6
- MQSeries for Windows Version 2 Release 0

MQCMDL_LEVEL_101

MQSeries for Windows Version 2 Release 0.1.

MQCMDL_LEVEL_110

MQSeries for Windows Version 2 Release 1.

MQCMDL_LEVEL_114

MQSeries for Version 1 Release 1.4.

MQCMDL_LEVEL_120

MQSeries for Version 1 Release 2.0.

MQCMDL_LEVEL_200

MQSeries for Windows NT[®] Version 2 Release 0.

MQCMDL_LEVEL_201

MQSeries for OS/2 Version 2 Release 0.1.

MQCMDL_LEVEL_210

MQSeries for OS/390 Version 2 Release 1.0.

MQCMDL_LEVEL_220

Level 220 of system control commands.

This value is returned by the following:

- MQSeries for AT&T GIS UNIX Version 2 Release 2
- MQSeries for SINIX and DC/OSx Version 2 Release 2
- MQSeries for SunOS Version 2 Release 2
- MQSeries for Tandem NonStop Kernel Version 2 Release 2

MQCMDL_LEVEL_221

Level 221 of system control commands.

This value is returned by the following:

- MQSeries for AIX Version 2 Release 2.1
- MQSeries for Digital OpenVMS Version 2 Release 2.1

MQCMDL_LEVEL_320

Level 320 of system control commands.

This value is returned by the following:

- MQSeries for OS/400
 - Version 3 Release 2
 - Version 3 Release 7

MQCMDL_LEVEL_420

Level 420 of system control commands.

This value is returned by the following:

- MQSeries for i5/OS
 - Version 4 Release 2.0
 - Version 4 Release 2.1

MQCMDL_LEVEL_500

Level 500 of system control commands.

This value is returned by the following:

- MQSeries for AIX Version 5 Release 0
- MQSeries for HP-UX Version 5 Release 0
- MQSeries for OS/2 Version 5 Release 0
- MQSeries for Solaris Version 5 Release 0
- MQSeries for Windows NT Version 5 Release 0

MQCMDL_LEVEL_510

Level 510 of system control commands.

This value is returned by the following:

- MQSeries for AIX Version 5 Release 1
- MQSeries for AS/400® Version 5 Release 1
- MQSeries for HP-UX Version 5 Release 1
- MQSeries for OS/2 Version 5 Release 1
- MQSeries for Compaq OpenVMS Alpha Version 5 Release 1
- MQSeries for Compaq NonStop Kernel Version 5 Release 1
- MQSeries for Compaq Tru64 UNIX Version 5 Release 1
- MQSeries for Solaris Version 5 Release 1
- MQSeries for Windows NT Version 5 Release 1

MQCMDL_LEVEL_520

Level 520 of system control commands.

This value is returned by the following:

- MQSeries for AIX Version 5 Release 2
- MQSeries for AS/400 Version 5 Release 2
- MQSeries for HP-UX Version 5 Release 2
- MQSeries for Linux Version 5 Release 2
- MQSeries for OS/390 Version 5 Release 2
- MQSeries for Sun Solaris Version 5 Release 2
- MQSeries for Windows NT Version 5 Release 2

MQCMDL_LEVEL_530

Level 530 of system control commands.

This value is returned by the following:

- Websphere MQ for AIX Version 5 Release 3
- Websphere MQ for HP-UX Version 5 Release 3
- Websphere MQ for i/Series Version 5 Release 3
- Websphere MQ for Linux for Intel Version 5 Release 3
- Websphere MQ for Linux for zSeries® Version 5 Release 3
- Websphere MQ for Solaris Version 5 Release 3
- Websphere MQ for Windows Version 5 Release 3
- Websphere MQ for z/OS Version 5 Release 3

MQCMDL_LEVEL_600

Level 600 of system control commands.

This value is returned by the following:

- Websphere MQ for AIX V6.0
- Websphere MQ for HP-UX V6.0
- Websphere MQ for i/Series V6.0
- Websphere MQ for Linux V6.0
- Websphere MQ for Solaris V6.0
- Websphere MQ for Windows V6.0
- Websphere MQ for z/OS V6.0

MQCMDL_LEVEL_700

Level 700 of system control commands.

This value is returned by the following:

- Websphere MQ for AIX V7.0
- Websphere MQ for HP-UX V7.0
- Websphere MQ for i5/OS V7.0
- WebSphere MQ for Linux V7.0
- Websphere MQ for Solaris V7.0
- Websphere MQ for Windows V7.0
- Websphere MQ for z/OS V7.0

The set of system control commands that corresponds to a particular value of the *CommandLevel* attribute varies according to the value of the *Platform* attribute; both must be used to decide which system control commands are supported.

To determine the value of this attribute, use the MQIA_COMMAND_LEVEL selector with the MQINQ call.

CommandServerControl (MQLONG)

Specifies whether the command server is to be started when the queue manager starts.

The value can be:

MQSVC_CONTROL_MANUAL

The command server is not to be started automatically.

MQSVC_CONTROL_Q_MGR

The command server is to be started automatically when the queue manager starts.

This attribute is not supported on z/OS.

To determine the value of this attribute, use the MQIA_CMD_SERVER_CONTROL selector with the MQINQ call.

ConfigurationEvent (MQLONG)

Controls whether configuration events are generated. This parameter applies to z/OS only.

To determine the value of this attribute, use the MQIA_CONFIGURATION_EVENT selector with the MQINQ call.

The value can be:

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

DeadLetterQName (MQCHAR48)

This is the name of a queue defined on the local queue manager as the dead-letter (undelivered-message) queue. Messages are sent to this queue if they cannot be routed to their correct destination.

For example, messages are put on this queue when:

- A message arrives at a queue manager, destined for a queue that is not yet defined on that queue manager
- A message arrives at a queue manager, but the queue for which it is destined cannot receive it because, possibly:
 - The queue is full
 - Put requests are inhibited
 - The sending node does not have authority to put messages on the queue

Applications can also put messages on the dead-letter queue.

Report messages are treated in the same way as ordinary messages; if the report message cannot be delivered to its destination queue (usually the queue specified by the *ReplyToQ* field in the message descriptor of the original message), the report message is placed on the dead-letter (undelivered-message) queue.

Note: Messages that have passed their expiry time (see MQMD - Expiry field) are **not** transferred to this queue when they are discarded. However, an expiration report message (MQRO_EXPIRATION) is still generated and sent to the *ReplyToQ* queue, if requested by the sending application.

Messages are not put on the dead-letter (undelivered-message) queue when the application that issued the put request has been notified synchronously of the problem by means of the reason code returned by the MQPUT or MQPUT1 call (for example, a message put on a local queue for which put requests are inhibited).

Messages on the dead-letter (undelivered-message) queue sometimes have their application message data prefixed with an MQDLH structure. This structure contains extra information that indicates why the message was placed on the dead-letter (undelivered-message) queue. See “MQDLH – Dead-letter header” on page 106 for more details of this structure.

This queue must be a local queue, with a *Usage* attribute of MQUS_NORMAL.

If a queue manager does not support a dead-letter (undelivered-message) queue, or one has not been defined, the name is all blanks. All WebSphere MQ queue managers support a dead-letter (undelivered-message) queue, but by default it is not defined.

If the dead-letter (undelivered-message) queue is not defined, full, or unusable for some other reason, a message which would have been transferred to it by a message channel agent is retained instead on the transmission queue.

To determine the value of this attribute, use the MQCA_DEAD_LETTER_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

DefXmitQName (MQCHAR48)

This is the name of the transmission queue that is used for the transmission of messages to remote queue managers, if there is no other indication of which transmission queue to use.

If there is no default transmission queue, the name is entirely blank. The initial value of this attribute is blank.

To determine the value of this attribute, use the MQCA_DEF_XMIT_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

DistLists (MQLONG)

This indicates whether the local queue manager supports distribution lists on the MQPUT and MQPUT1 calls. The value is one of the following:

MQDL_SUPPORTED

Distribution lists supported.

MQDL_NOT_SUPPORTED

Distribution lists not supported.

To determine the value of this attribute, use the MQIA_DIST_LISTS selector with the MQINQ call.

DNSGroup (MQCHAR18)

This is the name of the group for the TCP listener that handles inbound transmissions for the queue-sharing group to join when using Workload Manager Dynamic Domain Name Services support. The maximum length is 18 characters. If you leave this name blank, the queue-sharing group name is used.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQCA_DNS_GROUP selector with the MQINQ call. The length of this attribute is given by MQ_DNS_GROUP_NAME_LENGTH.

DNSWLM (MQLONG)

This specifies whether the TCP listener that handles inbound transmissions for the queue-sharing group registers with Workload Manager for Dynamic Domain Name Services

The value is one of the following:

MQDNSWLM_YES

The listener registers with Workload Manager.

MQDNSWLM_NO

The listener does not register with Workload Manager. This is the default value.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_DNS_WLM selector with the MQINQ call.

ExpiryInterval (MQLONG)

This indicates the frequency with which the queue manager scans the queues looking for expired messages. It is either a time interval in seconds in the range 1 through 99 999 999, or the following special value:

MQEXPI_OFF

The queue manager does not scan the queues looking for expired messages.

To determine the value of this attribute, use the MQIA_EXPIRY_INTERVAL selector with the MQINQ call.

This attribute is supported only on z/OS.

IGQPutAuthority (MQLONG)

This attribute applies only if the local queue manager is a member of a queue-sharing group. It indicates the type of authority checking that is performed when the local intra-group queuing agent (IGQ agent) removes a message from the shared transmission queue and places the message on a local queue. The value is one of the following:

MQIGQPA_DEFAULT

The user identifier checked for authorization is the value of the *UserIdentifier* field in the *separate* MQMD that is associated with the message when the message is on the shared transmission queue. This is the user identifier of the program that placed the message on the shared transmission queue, and is usually the same as the user identifier under which the remote queue manager is running.

If the RESLEVEL profile indicates that more than one user identifier is to be checked, the user identifier of the local IGQ agent (*IGQUserId*) is also checked.

MQIGQPA_CONTEXT

The user identifier checked for authorization is the value of the *UserIdentifier* field in the *separate* MQMD that is associated with the message when the message is on the shared transmission queue. This is the user identifier of the program that placed the message on the shared transmission queue, and is usually the same as the user identifier under which the remote queue manager is running.

If the RESLEVEL profile indicates that more than one user identifier is to be checked, the user identifier of the local IGQ agent (*IGQUserId*) and the value of the *UserIdentifier* field in the *embedded* MQMD are also checked. The latter user identifier is usually the user identifier of the application that originated the message.

MQIGQPA_ONLY_IGQ

The user identifier checked for authorization is the user identifier of the local IGQ agent (*IGQUserId*).

If the RESLEVEL profile indicates that more than one user identifier is to be checked, this user identifier is used for all checks.

MQIGQPA_ALTERNATE_OR_IGQ

The user identifier checked for authorization is the user identifier of the local IGQ agent (*IGQUserId*).

If the RESLEVEL profile indicates that more than one user identifier is to be checked, the value of the *UserIdentifier* field in the *embedded* MQMD is also checked. This user identifier is usually the user identifier of the application that originated the message.

To determine the value of this attribute, use the MQIA_IGQ_PUT_AUTHORITY selector with the MQINQ call.

This attribute is supported only on z/OS.

IGQUserId (MQLONG)

This attribute is applicable only if the local queue manager is a member of a queue-sharing group. It specifies the user identifier that is associated with the local intra-group queuing agent (IGQ agent). This identifier is one of the user identifiers that can be checked for authorization when the IGQ agent puts messages on local queues. The actual user identifiers checked depend on the setting of the *IGQPutAuthority* attribute, and on external security options.

If *IGQUserId* is blank, no user identifier is associated with the IGQ agent and the corresponding authorization check is not performed (although other user identifiers might still be checked for authorization).

To determine the value of this attribute, use the MQCA_IGQ_USER_ID selector with the MQINQ call. The length of this attribute is given by MQ_USER_ID_LENGTH.

This attribute is supported only on z/OS.

InhibitEvent (MQLONG)

This controls whether inhibit (Inhibit Get and Inhibit Put) events are generated. The value is one of the following:

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

For more information about events, see *WebSphere MQ Monitoring*.

To determine the value of this attribute, use the MQIA_INHIBIT_EVENT selector with the MQINQ call.

On z/OS, you cannot use the MQINQ call to determine the value of this attribute.

IntraGroupQueuing (MQLONG)

This attribute applies only if the local queue manager is a member of a queue-sharing group. It indicates whether intra-group queuing is enabled for the queue-sharing group. The value is one of the following:

MQIGQ_DISABLED

All messages destined for other queue managers in the queue-sharing group are transmitted using conventional channels..

MQIGQ_ENABLED

Messages destined for other queue managers in the queue-sharing group are transmitted using the shared transmission queue if the following condition is satisfied:

- The length of the message data plus transmission header does not exceed 63 KB (64 512 bytes).

It is recommended that somewhat more space than the size of MQXQH be allocated for the transmission header; the constant MQ_MSG_HEADER_LENGTH is provided for this purpose.

If this condition is not satisfied, the message is transmitted using conventional channels.

Note: When intra-group queuing is enabled, the order of messages transmitted using the shared transmission queue is not preserved relative to those transmitted using conventional channels.

To determine the value of this attribute, use the MQIA_INTRA_GROUP_QUEUING selector with the MQINQ call.

This attribute is supported only on z/OS.

IPAddressVersion (MQLONG)

Specifies which IP address version, either IPv4 or IPv6, is used.

This attribute is only relevant for systems that run both IPv4 and IPv6 and only affects channels defined as having a *TransportType* of MQXPY_TCP when one of the following conditions is true:

- The channel's *ConnectionName* is a host name that resolves to both an IPv4 and IPv6 address and its *LocalAddress* parameter is not specified.
- The channel's *ConnectionName* and *LocalAddress* are both host names that resolve to both IPv4 and IPv6 addresses.

The value can be:

MQIPADDR_IPV4
IPv4 is used.

MQIPADDR_IPV6
IPv6 is used.

To determine the value of this attribute, use the MQIA_IP_ADDRESS_VERSION selector with the MQINQ call.

ListenerTimer (MQLONG)

This is the time interval (in seconds) between WebSphere MQ attempts to restart the listener if there has been an APPC or TCP/IP failure. The value must be between 5 and 9999, with a default value of 60.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_LISTENER_TIMER selector with the MQINQ call.

LocalEvent (MQLONG)

This controls whether local error events are generated. The value is one of the following:

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

For more information about events, see *WebSphere MQ Monitoring*.

To determine the value of this attribute, use the MQIA_LOCAL_EVENT selector with the MQINQ call.

On z/OS, you cannot use the MQINQ call to determine the value of this attribute.

LoggerEvent (MQLONG)

This controls whether recovery log events are generated. The value is one of the following:

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

For more information about events, see *WebSphere MQ Monitoring*.

To determine the value of this attribute, use the MQIA_LOGGER_EVENT selector with the MQINQ call.

This attribute is supported only on AIX, HP-UX, i5/OS, Linux, Solaris, and Windows.

LUGroupName (MQCHAR8)

This is the generic LU name for the LU 6.2 listener that handles inbound transmissions for the queue-sharing group. If you leave this name blank, you cannot use this listener.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQCA_LU_GROUP_NAME selector with the MQINQ call. The length of this attribute is given by MQ_LU_NAME_LENGTH.

LUName (MQCHAR8)

This is the name of the LU to use for outbound LU 6.2 transmissions. Set this to the same LU that the listener uses for inbound transmissions. If you leave this name blank, the APPC/MVS default LU is used; this is variable, so always set LUName if you are using LU6.2.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQCA_LU_NAME selector with the MQINQ call. The length of this attribute is given by MQ_LU_NAME_LENGTH.

LU62ARMSuffix (MQCHAR2)

This is the suffix of the SYS1.PARMLIB member APPCPMxx, that nominates the LUADD for this channel initiator. The z/OS command SET APPC=xx is issued when ARM restarts the channel initiator. If you leave this name is blank, no SET APPC=xx is issued.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQCA_LU62_ARM_SUFFIX selector with the MQINQ call. The length of this attribute is given by MQ_ARM_SUFFIX_LENGTH.

LU62Channels (MQLONG)

This is the maximum number of channels that can be current, or clients that can be connected, that use the LU 6.2 transmission protocol. The value must be between 0 and 9999, with a default value of 200. If you set this to zero, the LU 6.2 transmission protocol is not used.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_LU62_CHANNELS selector with the MQINQ call.

MaxActiveChannels (MQLONG)

This is the maximum number of channels that can be active at any time. The value must be between 1 and 9999, with a default value of 200.

To determine the value of this attribute, use the MQIA_ACTIVE_CHANNELS selector with the MQINQ call.

MaxChannels (MQLONG)

This is the maximum number of channels that can be current (including server-connection channels with connected clients). The value must be between 1 and 9999 and has a default value of 200. It is possible that a system which is busy serving connections from the network could need a higher number than the default setting. You should determine the value that is correct for your environment, ideally by observing the behavior of your system during testing.

To determine the value of this attribute, use the MQIA_MAX_CHANNELS selector with the MQINQ call.

MaxHandles (MQLONG)

This is the maximum number of open handles that any one task can use concurrently. Each successful MQOPEN call for a single queue (or for an object that is not a queue) uses one handle. That handle becomes available for reuse when the object is closed. However, when a distribution list is opened, each queue in the distribution list is allocated a separate handle, and so that MQOPEN call uses as many handles as there are queues in the distribution list. This must be taken into account when deciding on a suitable value for *MaxHandles*.

The MQPUT1 call performs an MQOPEN call as part of its processing; as a result, MQPUT1 uses as many handles as MQOPEN would, but the handles are used only for the duration of the MQPUT1 call itself.

On z/OS, *task* means a CICS task, an MVS task, or an IMS dependent region.

The value is in the range 1 through 999 999 999. The default value is determined by the environment:

- On z/OS, the default value is 100.
- In all other environments, the default value is 256.

To determine the value of this attribute, use the MQIA_MAX_HANDLES selector with the MQINQ call.

MaxMsgLength (MQLONG)

This is the length of the longest *physical* message that the queue manager can handle. However, because the *MaxMsgLength* queue-manager attribute can be set independently of the *MaxMsgLength* queue attribute, the longest physical message that can be placed on a queue is the lesser of those two values.

If the queue manager supports segmentation, an application can put a *logical* message that is longer than the lesser of the two *MaxMsgLength* attributes, but only if the application specifies the MQMF_SEGMENTATION_ALLOWED flag in MQMD. If that flag is specified, the upper limit for the length of a logical message is 999 999 999 bytes, but usually resource constraints imposed by the operating system, or by the environment in which the application is running, result in a lower limit.

The lower limit for the *MaxMsgLength* attribute is 32 KB (32 768 bytes). The upper limit is 100 MB (104 857 600 bytes).

To determine the value of this attribute, use the MQIA_MAX_MSG_LENGTH selector with the MQINQ call.

MaxPriority (MQLONG)

This is the maximum message priority supported by the queue manager. Priorities range from zero (lowest) to *MaxPriority* (highest).

To determine the value of this attribute, use the MQIA_MAX_PRIORITY selector with the MQINQ call.

MaxUncommittedMsgs (MQLONG)

This is the maximum number of uncommitted messages that can exist within a unit of work. The number of uncommitted messages is the sum of the following since the start of the current unit of work:

- Messages put by the application with the MQPMO_SYNCPOINT option
- Messages retrieved by the application with the MQGMO_SYNCPOINT option
- Trigger messages and COA report messages generated by the queue manager for messages put with the MQPMO_SYNCPOINT option
- COD report messages generated by the queue manager for messages retrieved with the MQGMO_SYNCPOINT option

The following are *not* counted as uncommitted messages:

- Messages put or retrieved by the application outside a unit of work
- Trigger messages or COA/COD report messages generated by the queue manager as a result of messages put or retrieved outside a unit of work
- Expiration report messages generated by the queue manager (even if the call causing the expiration report message specified MQGMO_SYNCPOINT)
- Event messages generated by the queue manager (even if the call causing the event message specified MQPMO_SYNCPOINT or MQGMO_SYNCPOINT)

Note:

1. Exception report messages are generated by the Message Channel Agent (MCA), or by the application, and are treated in the same way as ordinary messages put or retrieved by the application.
2. When a message or segment is put with the MQPMO_SYNCPOINT option, the number of uncommitted messages is incremented by one regardless of how many physical messages actually result from the put. (More than one physical message might result if the queue manager needs to subdivide the message or segment.)
3. When a distribution list is put with the MQPMO_SYNCPOINT option, the number of uncommitted messages is incremented by one *for each physical message that is generated*. This can be as small as one, or as great as the number of destinations in the distribution list.

The lower limit for this attribute is 1; the upper limit is 999 999 999.

To determine the value of this attribute, use the MQIA_MAX_UNCOMMITTED_MSGS selector with the MQINQ call.

MQIAccounting (MQLONG)

This controls the collection of accounting information for MQI data.

The value is one of the following:

MQMON_ON

Collect API accounting data.

MQMON_OFF

Do not collect API accounting data. This is the default value.

If you set the queue manager attribute ACCTCONO to ENABLED, this value might be overridden for individual connections using the Options field in the MQCNO structure. Changes to this value are only effective for connections to the queue manager that occur after the change to the attribute.

This attribute is supported only on i5/OS, UNIX systems, and Windows.

To determine the value of this attribute, use the MQIA_ACCOUNTING_MQI selector with the MQINQ call.

MQIStatistics (MQLONG)

This controls the collection of statistics monitoring information for the queue manager.

The value is one of the following:

MQMON_ON

Collect MQI statistics.

MQMON_OFF

Do not collect MQI statistics. This is the default value.

This attribute is supported only on i5/OS, UNIX systems, and Windows.

To determine the value of this attribute, use the MQIA_STATISTICS_MQI selector with the MQINQ call.

MsgMarkBrowseInterval (MQLONG)

Time interval in milliseconds after which the queue manager can automatically remove the mark from browse messages.

This is a time interval (in milliseconds) after which the queue manager can automatically remove the mark from browse messages.

This attribute describes the time interval for which messages that have been marked as browsed by a call to MQGET, using the get message option MQGMO_MARK_BROWSE_CO_OP, are expected to remain mark as browsed.

The queue manager might automatically unmark browsed messages that have been marked as browsed for the cooperating set of handles when they have been marked for more than this approximate interval.

This does not affect the state of any message marked as browse, that was obtained by a call to MQGET, using the get message option MQGMO_MARK_BROWSE_HANDLE.

The value is not less than -1 and not greater than 999 999 999. The default value is 5000. A *MsgMarkBrowseInterval* of -1 represents an unlimited time interval. A *MsgMarkBrowseInterval* of 0 causes the queue manager to unmark the message immediately.

To determine the value of this attribute, use the MQIA_MSG_MARK_BROWSE_INTERVAL selector with the MQINQ call.

OutboundPortMax (MQLONG)

This is the highest port number in the range, defined by OutboundPortMin and OutboundPortMax, of port numbers to be used to bind outgoing channels . The value is an integer between 0 and 65535, and must be equal to or greater than the OutboundPortMin value. The default value is 0.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_OUTBOUND_PORT_MAX selector with the MQINQ call.

OutboundPortMin (MQLONG)

This is the lowest port number in the range, defined by OutboundPortMin and OutboundPortMax, of port numbers to be used to bind outgoing channels . The value is an integer between 0 and 65535, and must be equal to or less than the OutboundPortMax value. The default value is 0.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_OUTBOUND_PORT_MIN selector with the MQINQ call.

PerformanceEvent (MQLONG)

This controls whether performance-related events are generated. The value is one of the following:

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

For more information about events, see *WebSphere MQ Monitoring*.

To determine the value of this attribute, use the MQIA_PERFORMANCE_EVENT selector with the MQINQ call.

Platform (MQLONG)

This indicates the operating system on which the queue manager is running:

MQPL_AIX

AIX (same value as MQPL_UNIX).

MQPL_MVS

z/OS (same value as MQPL_ZOS).

MQPL_NSK

Compaq NonStop Kernel.

MQPL_OS2

OS/2.

MQPL_OS390

z/OS (same value as MQPL_ZOS).

MQPL_OS400

i5/OS.

MQPL_UNIX

UNIX systems.

MQPL_VMS

HP OpenVMS.

MQPL_WINDOWS_NT

Windows systems.

MQPL_ZOS

z/OS.

To determine the value of this attribute, use the MQIA_PLATFORM selector with the MQINQ call.

PropertyControl (MQLONG)

Specifies how message properties are handled for messages that are retrieved from queues using the MQGET call with the MQGMO_PROPERTIES_AS_Q_DEF option.

Local	Model	Alias	Remote	Cluster
X	X	X		

The value is one of the following:

MQPROP_ALL

All properties of the message are included with the message when it is delivered to the application. The properties, except those in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

MQPROP_COMPATIBILITY

If the message contains a property with a prefix of mcd., jms., usr. or mqext., all message properties are delivered to the application in an MQRFH2 header. Otherwise all properties of the message, except those contained in the message descriptor (or extension), are discarded and are no longer accessible to the application. This is the default value; it allows applications which expect JMS related properties to be in an MQRFH2 header in the message data to continue to work unmodified.

MQPROP_FORCE_MQRFH2

Properties are always returned in the message data in an MQRFH2 header regardless of whether the application specifies a message handle. A valid message handle supplied in the MsgHandle field of the MQGMO structure on the MQGET call is ignored. Properties of the message are not accessible via the message handle.

MQPROP_NONE

All properties of the message, except those in the message descriptor (or extension), are removed from the message before the message is delivered to the application.

This parameter is applicable to Local, Alias and Model queues. To determine its value, use the MQIA_PROPERTY_CONTROL selector with the MQINQ call.

PubSubNPInputMsg (MQLONG)

Whether to discard or keep an undelivered input message.

The value is one of the following:

MQUNDELIVERED_DISCARD

Non-persistent input messages may be discarded if they cannot be processed.

This is the default value.

MQUNDELIVERED_KEEP

Non-persistent input messages will not be discarded if they cannot be processed. In this situation the queued publish/subscribe interface will continue to retry the process at appropriate intervals and does not continue processing subsequent messages.

To determine the value of this attribute, use the MQIA_PUBSUB_NP_MSG selector with the MQINQ call.

PubSubNPResponse (MQLONG)

Controls the behavior of undelivered response messages.

The value is one of the following:

MQUNDELIVERED_NORMAL

Non-persistent responses which cannot be placed on the reply queue are put on the dead letter queue, if they cannot be placed on the DLQ then they are discarded.

MQUNDELIVERED_SAFE

Non-persistent responses which cannot be placed on the reply queue are put on the dead letter queue. If the response cannot be set and cannot be placed on the DLQ then the queued publish/subscribe interface will roll back the current operation and then retry at appropriate intervals and does not continue processing subsequent messages.

MQUNDELIVERED_DISCARD

Non-persistent responses are not placed on the reply queue are discarded.

This is the default value for new queue managers.

MQUNDELIVERED_KEEP

Non-persistent responses are not placed on the dead letter queue or discarded. Instead, the queued publish/subscribe interface will back out the current operation and then retry it at appropriate intervals.

To determine the value of this attribute, use the MQIA_PUBSUB_NP_RESP selector with the MQINQ call.

Default value for migrated queue managers.

If the queue manager has been migrated from Websphere MQ V6.0, the initial value of this attribute depends on the values of DiscardNonPersistentResponse and DLQNonPersistentResponse before migration, as shown in the following table.

		DLQNonPersistentResponse		
		Yes	No	Not set
DiscardNonPersistentResponse	Yes	MQUNDELIVERED_NORMAL	MQUNDELIVERED_DISCARD	MQUNDELIVERED_NORMAL
	No	MQUNDELIVERED_SAFE	MQUNDELIVERED_KEEP	MQUNDELIVERED_SAFE
	Not set	If SyncPointPersistent = No, MQUNDELIVERED_SAFE else MQUNDELIVERED_NORMAL	If SyncPointPersistent = No, MQUNDELIVERED_KEEP else MQUNDELIVERED_DISCARD	If SyncPointPersistent = No, MQUNDELIVERED_SAFE else MQUNDELIVERED_NORMAL

PubSubMaxMsgRetryCount (MQLONG)

The number of retries when processing a failed command message under syncpoint.

The value is one of the following:

0 - 999 999 999

The default value is 5.

To determine the value of this attribute, use the MQIA_PUBSUB_MAXMSG_RETRY_COUNT selector with the MQINQ call.

PubSubSyncPoint (MQLONG)

Whether only persistent messages or all messages are processed under syncpoint.

The value is one of the following:

MQSYNCPOINT_IFPER

This makes the queued publish/subscribe interface receive non-persistent messages outside syncpoint. If the daemon receives a publication outside syncpoint, the daemon forwards the publication to subscribers known to it outside syncpoint.

This is the default value.

MQSYNCPOINT_YES

This makes the queued publish/subscribe interface receive all messages under syncpoint.

To determine the value of this attribute, use the MQIA_PUBSUB_SYNC_PT selector with the MQINQ call.

PubSubMode (MQLONG)

Whether the publish/subscribe engine and the queued publish/subscribe interface are running, therefore allowing applications to publish/subscribe by using the application programming interface and the queues that are being monitored by the queued publish/subscribe interface.

The value is one of the following:

MQPSM_COMPAT

The publish/subscribe engine is running. It is therefore possible to publish/subscribe by using the application programming interface. The queued publish/subscribe interface is not running, therefore any message that is put to the queues that are monitored by the queued publish/subscribe interface will not be acted on. This setting is used for compatibility with WebSphere Message Broker V6 or earlier versions using this queue manager, because it needs to read the same queues from which the queued publish/subscribe interface normally reads.

MQPSM_DISABLED

The publish/subscribe engine and the queued publish/subscribe interface are not running. It is therefore not possible to publish/subscribe by using the application programming interface. Any publish/subscribe messages that are put to the queues that are monitored by the queued publish/subscribe interface will not be acted on.

MQPSM_ENABLED

The publish/subscribe engine and the queued publish/subscribe interface are running. It is therefore possible to publish/subscribe by using the application programming interface and the queues that are being monitored by the queued publish/subscribe interface. This is the queue manager's initial default value.

To determine the value of this attribute, use the MQIA_PUBSUB_MODE selector with the MQINQ call.

QMGrDesc (MQCHAR64)

Use this field for a commentary describing the queue manager. The content of the field is of no significance to the queue manager, but the queue manager might 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 might be translated incorrectly if this field is sent to another queue manager.

- On z/OS, the default value is the product name and version number.
- In all other environments, the default value is blanks.

To determine the value of this attribute, use the MQCA_Q_MGR_DESC selector with the MQINQ call. The length of this attribute is given by MQ_Q_MGR_DESC_LENGTH.

QMGrIdentifier (MQCHAR48)

This is an internally-generated unique name for the queue manager.

To determine the value of this attribute, use the MQCA_Q_MGR_IDENTIFIER selector with the MQINQ call. The length of this attribute is given by MQ_Q_MGR_IDENTIFIER_LENGTH.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, i5/OS, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

QMGrName (MQCHAR48)

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 MQMD - MsgId field). Queue managers that can intercommunicate

must therefore have names that differ in the first 12 characters, in order for message identifiers to be unique in the queue-manager network.

On z/OS, the name is the same as the subsystem name, which is limited to 4 nonblank characters.

To determine the value of this attribute, use the MQCA_Q_MGR_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_MGR_NAME_LENGTH.

QSGName (MQCHAR4)

This is the name of the queue-sharing group to which the local queue manager belongs. If the local queue manager does not belong to a queue-sharing group, the name is blank.

To determine the value of this attribute, use the MQCA_QSG_NAME selector with the MQINQ call. The length of this attribute is given by MQ_QSG_NAME_LENGTH.

This attribute is supported only on z/OS.

QueueAccounting (MQLONG)

This controls the collection of accounting information for queues.

The value is one of the following:

MQMON_NONE

Do not collect accounting data for queues, regardless of the setting of the queue accounting attribute ACCTQ. This is the default value.

MQMON_OFF

Do not collect accounting data for queues that specify QMGR in the ACCTQ queue attribute.

MQMON_ON

Collect accounting data for queues that specify QMGR in the ACCTQ queue attribute.

Changes to this value are only effective for connections to the queue manager that occur after the change to the attribute.

To determine the value of this attribute, use the MQIA_ACCOUNTING_Q selector with the MQINQ call.

QueueMonitoring (MQLONG)

This specifies the default setting for online monitoring of queues.

If the *QueueMonitoring* queue attribute is set to MQMON_Q_MGR, this attribute specifies the value which is assumed by the channel. The value can be:

MQMON_OFF

Online monitoring data collection is turned off. This is the queue manager's initial default value.

MQMON_NONE

Online monitoring data collection is turned off for queues regardless of the setting of their *QueueMonitoring* attribute.

MQMON_LOW

Online monitoring data collection is turned on, with a low ratio of data collection.

MQMON_MEDIUM

Online monitoring data collection is turned on, with a moderate ratio of data collection.

MQMON_HIGH

Online monitoring data collection is turned on, with a high ratio of data collection.

To determine the value of this attribute, use the MQIA_MONITORING_Q selector with the MQINQ call.

QueueStatistics (MQLONG)

This controls the collection of statistics data for queues.

The value is one of the following:

MQMON_NONE

Do not collect queue statistics for queues, regardless of the setting of the *QueueStatistics* queue attribute. This is the default value.

MQMON_OFF

Do not collect statistics data for queues that specify Queue Manager in the *QueueStatistics* queue attribute.

MQMON_ON

Collect statistics data for queues that specify Queue Manager in the *QueueStatistics* queue attribute.

To determine the value of this attribute, use the MQIA_STATISTICS_Q selector with the MQINQ call.

ReceiveTimeout (MQLONG)

This specifies how long a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to the inactive state. It applies only to message channels, not to MQI channels.

Use this value as follows:

- To specify that this number is a multiplier, to apply to the negotiated HBINT value, to determine how long a channel waits, set *ReceiveTimeoutType* to MQRCVTIME_MULTIPLY. Specify a value of 0 or a value in the range 2 to 99.
- To specify that this number is a value, in seconds, to add to the negotiated HBINT value to determine how long a channel waits, set *ReceiveTimeoutType* to MQRCVTIME_ADD. Specify a value in the range 1 to 999999.
- To specify that this number is a value, in seconds, for the channel to wait, set *ReceiveTimeoutType* to MQRCVTIME_EQUAL. Specify a value in the range 0 to 999999.

The default value is 0.

To stop a channel timing out its wait to receive data from its partner, set `ReceiveTimeoutType` to `MQRCVTIME_MULTIPLY` or `MQRCVTIME_EQUAL`, and `ReceiveTimeout` to 0.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the `MQIA_RECEIVE_TIMEOUT` selector with the `MQINQ` call.

ReceiveTimeoutMin (MQLONG)

This is the minimum time, in seconds, that a TCP/IP channel waits to receive data, including heartbeats, from its partner, before returning to the inactive state. It applies only to message channels, not to MQI channels. The value must be between zero and 999999, with a default of 0.

If you use `ReceiveTimeoutType` to specify that the TCP/IP channel wait time is to be calculated relative to the negotiated value of `HBINT`, and the resultant value is less than the value of this parameter, this value is used instead.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the `MQIA_RECEIVE_TIMEOUT_MIN` selector with the `MQINQ` call.

ReceiveTimeoutType (MQLONG)

This is the qualifier, applied to `ReceiveTimeout` to define how long a TCP/IP channel waits to receive data, including heartbeats, from its partner, before returning to the inactive state. It applies only to message channels, not to MQI channels.

The value is one of the following:

MQRCVTIME_MULTIPLY

`ReceiveTimeout` is a multiplier to apply to the negotiated `HBINT` value to determine how long a channel waits. This is the default value.

MQRCVTIME_ADD

`ReceiveTimeout` is a value, in seconds, to add to the negotiated `HBINT` value to determine how long a channel waits.

MQRCVTIME_EQUAL

`ReceiveTimeout` is a value, in seconds, that the channel waits.

To stop a channel timing out its wait to receive data from its partner, set `ReceiveTimeoutType` to `MQRCVTIME_MULTIPLY` or `MQRCVTIME_EQUAL`, and `ReceiveTimeout` to 0.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the `MQIA_RECEIVE_TIMEOUT_TYPE` selector with the `MQINQ` call.

RemoteEvent (MQLONG)

This controls whether remote error events are generated. The value is one of the following:

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

For more information about events, see *WebSphere MQ Monitoring*.

To determine the value of this attribute, use the MQIA_REMOTE_EVENT selector with the MQINQ call.

RepositoryName (MQCHAR48)

This is the name of a cluster for which this queue manager provides a repository-manager service. If the queue manager provides this service for more than one cluster, *RepositoryNameList* specifies the name of a namelist object that identifies the clusters, and *RepositoryName* is blank. At least one of *RepositoryName* and *RepositoryNameList* must be blank.

This attribute is supported only on AIX, HP-UX, i5/OS, Linux, Solaris, Windows, and z/OS.

To determine the value of this attribute, use the MQCA_REPOSITORY_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_MGR_NAME_LENGTH.

RepositoryNameList (MQCHAR48)

This is the name of a namelist object that contains the names of clusters for which this queue manager provides a repository-manager service. If the queue manager provides this service for only one cluster, the namelist object contains only one name. Alternatively, *RepositoryName* can be used to specify the name of the cluster, in which case *RepositoryNameList* is blank. At least one of *RepositoryName* and *RepositoryNameList* must be blank.

This attribute is supported only on AIX, HP-UX, i5/OS, Linux, Solaris, Windows, and z/OS.

To determine the value of this attribute, use the MQCA_REPOSITORY_NAMELIST selector with the MQINQ call. The length of this attribute is given by MQ_NAMELIST_NAME_LENGTH.

ScyCase(MQCHAR8)

Specifies whether or not the queue manager supports security profile names in mixed case, or in uppercase only.

The value is one of the following:

MQSCYC_UPPER

Security profile names must be in uppercase.

MQSCYC_MIXED

Security profile names can be in uppercase or in mixed case.

Changes to this attribute take effect when a Refresh Security command is run with *SecurityType*(MQSECTYPE_CLASSES) specified.

This attribute is supported only on z/OS.

To determine the value of this attribute, use the MQIA_SECURITY_CASE selector with the MQINQ call.

SharedQMgrName (MQLONG)

This specifies whether the *ObjectQmgrName* should be used or treated as the local queue manager on an MQOPEN call, for a shared queue, when the *ObjectQmgrName* is that of another queue manager in the queue-sharing group.

The value can be:

MQSQQM_USE

ObjectQmgrName is used and the appropriate transmission queue is opened.

MQSQQM_IGNORE

If the target queue is shared, and the *ObjectQmgrName* is that of a queue manager in the same queue-sharing group, the open is performed locally.

This attribute is valid only on z/OS.

To determine the value of this attribute, use the MQIA_SHARED_Q_Q_MGR_NAME selector with the MQINQ call.

SSLEvent (MQLONG)

This specifies whether SSL events are generated.

The value is one of the following:

MQEVR_ENABLED

Generate SSL events, as follows:
MQRC_CHANNEL_SSL_ERROR

MQEVR_DISABLED

Do not generate SSL events; this is the default value.

To determine the value of this attribute, use the MQIA_SSL_EVENT selector with the MQINQ call.

SSLFIPSRequired (MQLONG)

This lets you specify that only FIPS-certified algorithms are to be used if the cryptography is executed in WebSphere MQ-provided software. If cryptographic hardware is configured, the cryptography modules used are those provided by the hardware product; these may or may not be FIPS-certified to a particular level depending on the hardware product in use.

The value is one of the following:

MQSSL_FIPS_NO

Use any CipherSpec supported on the platform in use. This is the default value.

MQSSL_FIPS_YES

Use only FIPS-certified cryptographic algorithms in the CipherSpecs allowed on all SSL connections from and to this queue manager.

This parameter is valid only on UNIX platforms and Windows.

To determine the value of this attribute, use the MQIA_SSL_FIPS_REQUIRED selector with the MQINQ call.

SSLKeyResetCount (MQLONG)

This specifies when SSL channel message channel agents (MCAs) that initiate communication reset the secret key used for encryption on the channel. The value represents the total number of unencrypted bytes that are sent and received on the channel before the secret key is renegotiated. The number of bytes includes control information sent by the MCA.

The value is a number between 0 and 999 999 999, with a default value of 0. If you specify an SSL/TLS secret key reset count between 1 byte and 32Kb, SSL/TLS channels will use a secret key reset count of 32Kb. This is to avoid the overhead of excessive key resets which would occur for small SSL/TLS secret key reset values.

The secret key is renegotiated when the total number of unencrypted bytes sent and received by the initiating channel MCA exceeds the specified value, or if channel heartbeats are enabled before data is sent or received following a channel heartbeat, whichever occurs first.

The count of bytes sent and received for renegotiation includes control information sent and received by the channel MCA and is reset whenever a renegotiation occurs.

Use a value of 0 to indicate that secret keys are never renegotiated.

To determine the value of this attribute, use the MQIA_SSL_RESET_COUNT selector with the MQINQ call.

StartStopEvent (MQLONG)

This controls whether start and stop events are generated. The value is one of the following:

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

For more information about events, see *WebSphere MQ Monitoring*.

To determine the value of this attribute, use the MQIA_START_STOP_EVENT selector with the MQINQ call.

StatisticsInterval (MQLONG)

This specifies how often (in seconds) to write statistics monitoring data to the monitoring queue.

The value is an integer in the range 0 to 604800, with a default value of 1800 (30 minutes).

To determine the value of this attribute, use the MQIA_STATISTICS_INTERVAL selector with the MQINQ call.

SyncPoint (MQLONG)

This indicates whether the local queue manager supports units of work and syncpointing with the MQGET, MQPUT, and MQPUT1 calls.

MQSP_AVAILABLE

Units of work and syncpointing available.

MQSP_NOT_AVAILABLE

Units of work and syncpointing not available.

- On z/OS this value is never returned.

To determine the value of this attribute, use the MQIA_SYNCPOINT selector with the MQINQ call.

TCPChannels (MQLONG)

This is the maximum number of channels that can be current, or clients that can be connected, that use the TCP/IP transmission protocol. The value must be between 0 and 9999, with a default value of 200. If you specify 0, TCP/IP is not used.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_TCP_CHANNELS selector with the MQINQ call.

TCPKeepAlive (MQLONG)

This specifies whether to use TCP KEEPALIVE to check that the other end of the connection is still available. If it is not available, the channel is closed.

The value is one of the following:

MQTCPKEEP_YES

Use TCP KEEPALIVE as specified in the TCP profile configuration data set. If you specify the channel attribute KeepAliveInterval (KAINT), the value to which it is set is used.

MQTCPKEEP_NO

Do not use TCP KEEPALIVE. This is the default value.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_TCP_KEEP_ALIVE selector with the MQINQ call.

TCPName (MQCHAR8)

This is the name of either the only or default TCP/IP system that you are using, depending on the value of TCPStackType. The default value is TCPIP.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQCA_TCP_NAME selector with the MQINQ call. The length of this attribute is given by MQ_TCP_NAME_LENGTH.

TCPStackType (MQLONG)

This specifies whether the channel initiator can use only the TCP/IP address space specified in TCPName, or can optionally bind to any selected TCP/IP address

The value is one of the following:

MQTCPSTACK_SINGLE

The channel initiator can use only the TCP/IP address spaces named in TCPName. This is the default value.

MQTCPSTACK_MULTIPLE

The channel initiator can use any TCP/IP address space available to it. It defaults to the one specified in TCPName if no other is specified for a channel or listener.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_TCP_STACK_TYPE selector with the MQINQ call.

TraceRouteRecording (MQLONG)

This controls the recording of trace- route information.

The value is one of the following:

MQRECORDING_DISABLED

No appending to trace- route messages allowed.

MQRECORDING_Q

Put trace- route messages to fixed named queue.

MQRECORDING_MSG

Put trace- route messages to a queue determined using the message itself. This is the default value

To determine the value of this attribute, use the MQIA_TRACE_ROUTE_RECORDING selector with the MQINQ call.

TriggerInterval (MQLONG)

This is a time interval (in milliseconds) used to restrict the number of trigger messages. This is relevant only when the *TriggerType* is MQTT_FIRST. In this case trigger messages are usually generated only when a suitable message arrives on the queue, and the queue was previously empty. Under certain circumstances,

however, an additional trigger message can be generated with `MQTT_FIRST` triggering even if the queue was not empty. These additional trigger messages are not generated more often than every *TriggerInterval* milliseconds.

For more information on triggering, see the *WebSphere MQ Application Programming Guide*.

The value is not less than 0 and not greater than 999 999 999. The default value is 999 999 999.

To determine the value of this attribute, use the `MQIA_TRIGGER_INTERVAL` selector with the `MQINQ` call.

TriggerInterval (MQLONG)

This is a time interval (in milliseconds) used to restrict the number of trigger messages. This is relevant only when the *TriggerType* is `MQTT_FIRST`. In this case trigger messages are usually generated only when a suitable message arrives on the queue, and the queue was previously empty. Under certain circumstances, however, an additional trigger message can be generated with `MQTT_FIRST` triggering even if the queue was not empty. These additional trigger messages are not generated more often than every *TriggerInterval* milliseconds.

For more information on triggering, see the *WebSphere MQ Application Programming Guide*.

The value is not less than 0 and not greater than 999 999 999. The default value is 999 999 999.

To determine the value of this attribute, use the `MQIA_TRIGGER_INTERVAL` selector with the `MQINQ` call.

Attributes for queues

There are five types of queue definition. Some queue attributes apply to all types of queue; other queue attributes apply only to certain types of queue.

Types of queue: The queue manager supports the following types of queue definition:

Local queue

This is a physical queue that stores messages and can be one of the following:

Local queue manager queue

The queue exists on the local queue manager (private in z/OS parlance).

Shared queue (z/OS only)

This is a physical queue that stores messages. The queue exists in a shared repository that is accessible to all the queue managers that belong to the queue-sharing group that owns the shared repository.

Applications connected to any queue manager in the queue-sharing group can place messages on and remove messages from queues of this type. Such queues are effectively the same as local queues. The value of the *QType* queue attribute is `MQQT_LOCAL`.

Applications connected to the local queue manager can place messages on and remove messages from queues of this type. The value of the *QType* queue attribute is MQQT_LOCAL.

Cluster queue

This is a physical queue that stores messages. The queue exists either on the local queue manager, or on one or more of the queue managers that belong to the same cluster as the local queue manager.

Applications connected to the local queue manager can place messages on queues of this type, regardless of the location of the queue. If an instance of the queue exists on the local queue manager, the queue behaves in the same way as a local queue, and applications connected to the local queue manager can remove messages from the queue. The value of the *QType* queue attribute is MQQT_CLUSTER.

Remote queue

This is not a physical queue; it is the local definition of a queue that exists on a remote queue manager. The local definition of the remote queue contains information that tells the local queue manager how to route messages to the remote queue manager.

Applications connected to the local queue manager can place messages on queues of this type; the messages are placed on the local transmission queue used to route messages to the remote queue manager. Applications cannot remove messages from remote queues. The value of the *QType* queue attribute is MQQT_REMOTE.

You can also use a remote queue definition for:

- Reply-queue aliasing
In this case the name of the definition is the name of a reply-to queue. For more information, see *WebSphere MQ Intercommunications*.
- Queue-manager aliasing
In this case the name of the definition is an alias for a queue manager, and not the name of a queue. For more information, see *WebSphere MQ Intercommunications*.

Alias queue

This is not a physical queue; it is an alternative name for a local queue, a shared queue, a cluster queue, or a remote queue. The name of the queue to which the alias resolves is part of the definition of the alias queue.

Applications connected to the local queue manager can place messages on queues of this type; the messages are placed on the queue to which the alias resolves. Applications can remove messages from queues of this type if the alias resolves to a local queue, a shared queue, or a cluster queue that has a local instance. The value of the *QType* queue attribute is MQQT_ALIAS.

Model queue

This is not a physical queue; it is a set of queue attributes from which a local queue can be created.

Messages cannot be stored on queues of this type.

Queue attributes: Some queue attributes apply to all types of queue; other queue attributes apply only to certain types of queue. The types of queue to which an attribute applies are shown in Table 96 on page 617 and subsequent tables.

Table 96 summarizes the attributes that are specific to queues. The attributes are described in alphabetic order.

Note: The names of the attributes shown in this book are descriptive names used with the MQINQ and MQSET calls; the names are the same as for the PCF commands. When MQSC commands are used to define, alter, or display attributes, alternative short names are used; see the *WebSphere MQ Script (MQSC) Command Reference* for details.

Table 96. Attributes for queues. The columns apply as follows:

- The column for local queues applies also to shared queues.
- The column for model queues indicates which attributes are inherited by the local queue created from the model queue.
- The column for cluster queues indicates the attributes that can be inquired when the cluster queue is opened for inquire alone, or for inquire and output. If the cluster queue is opened for inquire plus one or more of input, browse, or set, the column for local queues applies instead.

Attribute	Description	Local	Model	Alias	Remote	Cluster	Topic
<i>AlterationDate</i>	Date when definition was last changed	X		X	X		AlterationDate
<i>AlterationTime</i>	Time when definition was last changed	X		X	X		AlterationTime
<i>BackoutRequeueQName</i>	Excessive backout requeue queue name	X	X				BackoutRequeueQName
<i>BackoutThreshold</i>	Backout threshold	X	X				BackoutThreshold
<i>BaseQName</i>	Queue name to which alias resolves			X			BaseQName
<i>CFStrucName</i>	Coupling-facility structure name	X	X				CFStrucName
<i>ClusterName</i>	Name of cluster to which queue belongs	X		X	X	X	ClusterName
<i>ClusterNameList</i>	Name of namelist object containing names of clusters to which queue belongs	X		X	X		ClusterNameList
<i>CLWLQueuePriority</i>	Cluster workload queue priority	X		X	X	X	CLWLQueuePriority
<i>CLWLQueueRank</i>	Cluster workload queue rank	X		X	X	X	CLWLQueueRank
<i>CLWLUseQ</i>	Use remote queue	X					CLWLUseQ
<i>CreationDate</i>	Date that the queue was created	X					CreationDate
<i>CreationTime</i>	Time that the queue was created	X					CreationTime
<i>CurrentQDepth</i>	Current queue depth	X					CurrentQDepth
<i>DefaultPutResponse</i>	Default put response	X	X	X	X		DefaultPutResponse
<i>DefBind</i>	Default binding	X		X	X	X	DefBind
<i>DefinitionType</i>	Queue definition type	X	X				DefinitionType attribute
<i>DefInputOpenOption</i>	Default input open option	X	X				DefInputOpenOption
<i>DefPersistence</i>	Default message persistence	X	X	X	X	X	DefPersistence
<i>DefPriority</i>	Default message priority	X	X	X	X	X	DefPriority
<i>DefReadAhead</i>	Default read ahead	X	X	X			DefReadAhead
<i>DistLists</i>	Distribution list support	X	X				DistLists

Table 96. Attributes for queues (continued). The columns apply as follows:

- The column for local queues applies also to shared queues.
- The column for model queues indicates which attributes are inherited by the local queue created from the model queue.
- The column for cluster queues indicates the attributes that can be inquired when the cluster queue is opened for inquire alone, or for inquire and output. If the cluster queue is opened for inquire plus one or more of input, browse, or set, the column for local queues applies instead.

Attribute	Description	Local	Model	Alias	Remote	Cluster	Topic
<i>HardenGetBackout</i>	Whether to maintain an accurate backout count	X	X				HardenGetBackout
<i>IndexType</i>	Index type	X	X				IndexType
<i>InhibitGet</i>	Whether get operations for the queue are allowed	X	X	X			InhibitGet
<i>InhibitPut</i>	Whether put operations for the queue are allowed	X	X	X	X	X	InhibitPut
<i>InitiationQName</i>	Name of initiation queue	X	X				InitiationQName
<i>MaxMsgLength</i>	Maximum message length in bytes	X	X				MaxMsgLength
<i>MaxQDepth</i>	Maximum queue depth	X	X				MaxQDepth
<i>MsgDeliverySequence</i>	Message delivery sequence	X	X				MsgDeliverySequence attribute
<i>NonPersistentMessageClass</i>	Reliability goal for non persistent messages	X	X				NonPersistentMessage Class
<i>OpenInputCount</i>	Number of opens for input	X					OpenInputCount
<i>OpenOutputCount</i>	Number of opens for output	X					OpenOutputCount
<i>PropertyControl</i>	Property control	X	X	X			PropertyControl
<i>ProcessName</i>	Process name	X	X				ProcessName
<i>QDepthHighEvent</i>	Whether Queue Depth High events are generated	X	X				QDepthHighEvent attribute
<i>QDepthHighLimit</i>	High limit for queue depth	X	X				QDepthHighLimit
<i>QDepthLowEvent</i>	Whether Queue Depth Low events are generated	X	X				QDepthLowEvent attribute
<i>QDepthLowLimit</i>	Low limit for queue depth	X	X				QDepthLowLimit attribute
<i>QDepthMaxEvent</i>	Whether Queue Full events are generated	X	X				QDepthMaxEvent
<i>QDesc</i>	Queue description	X	X	X	X	X	QDesc
<i>QName</i>	Queue name	X		X	X	X	QName
<i>QServiceInterval</i>	Target for queue service interval	X	X				QServiceInterval
<i>QServiceIntervalEvent</i>	Whether Service Interval High or Service Interval OK events are generated	X	X				QServiceIntervalEvent attribute
<i>QSGDisp</i>	Queue-sharing group disposition	X		X	X		QSGDisp attribute
<i>QueueAccounting</i>	Queue accounting data collection	X	X	X	X	X	QueueAccounting

Table 96. Attributes for queues (continued). The columns apply as follows:

- The column for local queues applies also to shared queues.
- The column for model queues indicates which attributes are inherited by the local queue created from the model queue.
- The column for cluster queues indicates the attributes that can be inquired when the cluster queue is opened for inquire alone, or for inquire and output. If the cluster queue is opened for inquire plus one or more of input, browse, or set, the column for local queues applies instead.

Attribute	Description	Local	Model	Alias	Remote	Cluster	Topic
<i>QueueMonitoring</i>	Online monitoring data for queues	X	X				QueueMonitoring
<i>QueueStatistics</i>	Queue statistics data collection	X	X	X	X	X	QueueStatistics
<i>QType</i>	Queue type	X		X	X	X	QType
<i>RemoteQMgrName</i>	Name of remote queue manager				X		RemoteQMgrName
<i>RemoteQName</i>	Name of remote queue				X		RemoteQName
<i>RetentionInterval</i>	Retention interval	X	X				RetentionInterval
<i>Scope</i>	Whether an entry for the queue also exists in a cell directory	X		X	X		Scope
<i>Shareability</i>	Queue shareability	X	X				Shareability
<i>StorageClass</i>	Storage class for queue	X	X				StorageClass
<i>TriggerControl</i>	Trigger control	X	X				TriggerControl
<i>TriggerData</i>	Trigger data	X	X				TriggerData
<i>TriggerDepth</i>	Trigger depth	X	X				TriggerDepth
<i>TriggerMsgPriority</i>	Threshold message priority for triggers	X	X				TriggerMsgPriority
<i>TriggerType</i>	Trigger type	X	X				TriggerType
<i>Usage</i>	Queue usage	X	X				Usage attribute
<i>XmitQName</i>	Transmission queue name				X		XmitQName

AlterationDate (MQCHAR12)

Local	Model	Alias	Remote	Cluster
X		X	X	

This is the date when the definition was last changed. The format of the date is YYYY-MM-DD, padded with two trailing blanks to make the length 12 bytes (for example, 1992-09-23**bb**, where **bb** represents 2 blank characters).

The values of certain attributes (for example, *CurrentQDepth*) change as the queue manager operates. Changes to these attributes do not affect *AlterationDate*. Also, changes resulting from use of the MQSET call do not affect *AlterationDate*.

To determine the value of this attribute, use the MQCA_ALTERATION_DATE selector with the MQINQ call. The length of this attribute is given by MQ_DATE_LENGTH.

AlterationTime (MQCHAR8)

Local	Model	Alias	Remote	Cluster
X		X	X	

This is the time when the definition was last changed. The format of the time is HH.MM.SS using the 24-hour clock, with a leading zero if the hour is less than 10 (for example 09.10.20).

- On z/OS, the time is Greenwich Mean Time (GMT), subject to the system clock being set accurately to GMT.
- In other environments, the time is local time.

The values of certain attributes (for example, *CurrentQDepth*) change as the queue manager operates. Changes to these attributes do not affect *AlterationTime*. Also, changes resulting from use of the MQSET call do not affect *AlterationTime*.

To determine the value of this attribute, use the MQCA_ALTERATION_TIME selector with the MQINQ call. The length of this attribute is given by MQ_TIME_LENGTH.

BackoutRequeueQName (MQCHAR48)

This is the 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.

Local	Model	Alias	Remote	Cluster
X	X			

WebSphere MQ classes for JMS uses this attribute to determine where to transfer a message that has already been backed out the maximum number of times as specified by the *BackoutThreshold* attribute.

To determine the value of this attribute, use the MQCA_BACKOUT_REQ_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

BackoutThreshold (MQLONG)

This is the backout threshold. Apart from allowing its value to be queried, the queue manager takes no action based on the value of this attribute.

Local	Model	Alias	Remote	Cluster
X	X			

WebSphere MQ classes for JMS uses this attribute to determine how many times to allow a message to be backed out before transferring the message to the queue specified by the *BackoutRequeueQName* attribute.

To determine the value of this attribute, use the MQIA_BACKOUT_THRESHOLD selector with the MQINQ call.

BaseQName (MQCHAR48)

This is the name of a queue that is defined to the local queue manager.

Local	Model	Alias	Remote	Cluster
		X		

(For more information on queue names, see MQOD - ObjectName field.) The queue is one of the following types:

MQQT_LOCAL

Local queue.

MQQT_REMOTE

Local definition of a remote queue.

MQQT_CLUSTER

Cluster queue.

To determine the value of this attribute, use the MQCA_BASE_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

BaseType (MQCFIN)

The type of object to which the alias resolves.

Local	Model	Alias	Remote	Cluster
		X		

The value is one of the following:

MQOT_Q

Base object type is a queue

MQOT_TOPIC

Base object type is a topic

CFStrucName (MQCHAR12)

This is the name of the coupling-facility structure where the messages on the queue are stored. The first character of the name is in the range A through Z, and the remaining characters are in the range A through Z, 0 through 9, or blank.

Local	Model	Alias	Remote	Cluster
X	X			

To get the full name of the structure in the coupling facility, suffix the value of the *QSGName* queue-manager attribute with the value of the *CFStrucName* queue attribute.

This attribute applies only to shared queues; it is ignored if *QSGDisp* does not have the value MQQSGD_SHARED.

To determine the value of this attribute, use the MQCA_CF_STRUC_NAME selector with the MQINQ call. The length of this attribute is given by MQ_CF_STRUC_NAME_LENGTH.

This attribute is supported only on z/OS.

ClusterName (MQCHAR48)

This is the name of the cluster to which the queue belongs.

Local	Model	Alias	Remote	Cluster
X		X	X	X

If the queue belongs to more than one cluster, *ClusterNameList* specifies the name of a namelist object that identifies the clusters, and *ClusterName* is blank. At least one of *ClusterName* and *ClusterNameList* must be blank.

To determine the value of this attribute, use the MQCA_CLUSTER_NAME selector with the MQINQ call. The length of this attribute is given by MQ_CLUSTER_NAME_LENGTH.

ClusterNameList (MQCHAR48)

This is the name of a namelist object that contains the names of clusters to which this queue belongs.

Local	Model	Alias	Remote	Cluster
X		X	X	

If the queue belongs to only one cluster, the namelist object contains only one name. Alternatively, *ClusterName* can be used to specify the name of the cluster, in which case *ClusterNameList* is blank. At least one of *ClusterName* and *ClusterNameList* must be blank.

To determine the value of this attribute, use the MQCA_CLUSTER_NAMELIST selector with the MQINQ call. The length of this attribute is given by MQ_NAMELIST_NAME_LENGTH.

CLWLQueuePriority (MQLONG)

This is the cluster workload queue priority, a value between 0 and 9 representing the priority of the queue.

Local	Model	Alias	Remote	Cluster
X		X	X	X

For more information, see *WebSphere MQ Queue Manager Clusters*.

To determine the value of this attribute, use the MQCA_CLWL_Q_PRIORITY selector with the MQINQ call. The length of this attribute is given by MQ_CLWL_Q_PRIORITY_LENGTH.

CLWLQueueRank (MQLONG)

This is the cluster workload queue rank, a value between 0 and 9 representing the rank of the queue.

Local	Model	Alias	Remote	Cluster
X		X	X	X

For more information, see *WebSphere MQ Queue Manager Clusters*.

To determine the value of this attribute, use the MQCA_CLWL_Q_RANK selector with the MQINQ call. The length of this attribute is given by MQ_CLWL_Q_RANK_LENGTH.

CLWLUseQ (MQLONG)

This defines the behavior of an MQPUT when the target queue has both a local instance and at least one remote cluster instance. If the put originates from a cluster channel, this attribute does not apply.

Local	Model	Alias	Remote	Cluster
X				

The value is one of the following:

MQCLWL_USEQ_ANY

Use remote and local queues.

MQCLWL_USEQ_LOCAL

Do not use remote queues.

MQCLWL_USEQ_AS_Q_MGR

Inherit definition from queue manager's MQIA_CLWL_USEQ.

For more information, see *WebSphere MQ Queue Manager Clusters*.

To determine the value of this attribute, use the MQCA_CLWL_USEQ selector with the MQINQ call. The length of this attribute is given by MQ_CLWL_USEQ_LENGTH.

CreationDate (MQCHAR12)

This is the date when the queue was created.

Local	Model	Alias	Remote	Cluster
X				

The format of the date is YYYY-MM-DD, padded with two trailing blanks to make the length 12 bytes (for example, 1992-09-23bb, where bb represents 2 blank characters).

- On i5/OS, the creation date of a queue can differ from that of the underlying operating system entity (file or userspace) that represents the queue.

To determine the value of this attribute, use the MQCA_CREATION_DATE selector with the MQINQ call. The length of this attribute is given by MQ_CREATION_DATE_LENGTH.

CreationTime (MQCHAR8)

This is the time when the queue was created.

Local	Model	Alias	Remote	Cluster
X				

The format of the time is HH.MM.SS using the 24-hour clock, with a leading zero if the hour is less than 10 (for example 09.10.20).

- On z/OS, the time is Greenwich Mean Time (GMT), subject to the system clock being set accurately to GMT.
- In other environments, the time is local time.
- On i5/OS, the creation time of a queue can differ from that of the underlying operating system entity (file or userspace) that represents the queue.

To determine the value of this attribute, use the MQCA_CREATION_TIME selector with the MQINQ call. The length of this attribute is given by MQ_CREATION_TIME_LENGTH.

CurrentQDepth (MQLONG)

This is the number of messages currently on the queue.

Local	Model	Alias	Remote	Cluster
X				

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 that 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 that have yet to be committed.

The count also includes messages that have passed their expiry time but have not yet been discarded, although these messages are not eligible to be retrieved. See MQMD - Expiry field for more information.

Unit-of-work processing and the segmentation of messages can both cause *CurrentQDepth* to exceed *MaxQDepth*. However, this does not affect the retrievability of the messages; *all* messages on the queue can be retrieved using the MQGET call in the normal way.

The value of this attribute fluctuates as the queue manager operates.

To determine the value of this attribute, use the MQIA_CURRENT_Q_DEPTH selector with the MQINQ call.

DefaultPutResponse (MQLONG)

Specifies the type of response to be used for put operations to the queue when an application specifies MQPMO_RESPONSE_AS_Q_DEF.

Local	Model	Alias	Remote	Cluster
X	X	X	X	

The value is one of the following:

| MQPRT_SYNC_RESPONSE

| The put operation is issued synchronously, returning a response.

| MQPRT_ASYNC_RESPONSE

| The put operation is issued asynchronously, returning a subset of MQMD fields.

DefBind (MQLONG)

This is the default binding that is used when MQOO_BIND_AS_Q_DEF is specified on the MQOPEN call and the queue is a cluster queue.

Local	Model	Alias	Remote	Cluster
X		X	X	X

The value is one of the following:

MQBND_BIND_ON_OPEN

Binding fixed by MQOPEN call.

MQBND_BIND_NOT_FIXED

Binding not fixed.

To determine the value of this attribute, use the MQIA_DEF_BIND selector with the MQINQ call.

DefinitionType (MQLONG)

This indicates how the queue was defined.

Local	Model	Alias	Remote	Cluster
X	X			

The value is one of the following:

MQQDT_PREDEFINED

The queue is a permanent queue created by the system administrator; only the system administrator can delete it.

Predefined queues are created using the DEFINE MQSC command, and can be deleted only by using the DELETE MQSC command. Predefined queues cannot be created from model queues.

Commands can be issued either by an operator, or by an authorized user sending a command message to the command input queue (see CommandInputQName attribute for more information).

MQQDT_PERMANENT_DYNAMIC

The queue is a permanent queue that was created by an application issuing an MQOPEN call with the name of a model queue specified in the object descriptor MQOD. The model queue definition had the value MQQDT_PERMANENT_DYNAMIC for the *DefinitionType* attribute.

This type of queue can be deleted using the MQCLOSE call. See “MQCLOSE – Close object” on page 422 for more details.

The value of the *QSGDisp* attribute for a permanent dynamic queue is MQQSGD_Q_MGR.

MQQDT_TEMPORARY_DYNAMIC

The queue is a temporary queue that was created by an application issuing an MQOPEN call with the name of a model queue specified in the object descriptor MQOD. The model queue definition had the value MQQDT_TEMPORARY_DYNAMIC for the *DefinitionType* attribute.

This type of queue is deleted automatically by the MQCLOSE call when it is closed by the application that created it.

The value of the *QSGDisp* attribute for a temporary dynamic queue is MQQSGD_Q_MGR.

MQQDT_SHARED_DYNAMIC

The queue is a shared permanent queue that was created by an application issuing an MQOPEN call with the name of a model queue specified in the object descriptor MQOD. The model queue definition had the value MQQDT_SHARED_DYNAMIC for the *DefinitionType* attribute.

This type of queue can be deleted using the MQCLOSE call. See “MQCLOSE – Close object” on page 422 for more details.

The value of the *QSGDisp* attribute for a shared dynamic queue is MQQSGD_SHARED.

This attribute in a model queue definition does not indicate how the model queue was defined, because model queues are always predefined. Instead, the value of this attribute in the model queue is used to determine the *DefinitionType* of each of the dynamic queues created from the model queue definition using the MQOPEN call.

To determine the value of this attribute, use the MQIA_DEFINITION_TYPE selector with the MQINQ call.

DefInputOpenOption (MQLONG)

This is the default way in which to open the queue for input.

Local	Model	Alias	Remote	Cluster
X	X			

It applies if the MQOO_INPUT_AS_Q_DEF option is specified on the MQOPEN call when the queue is opened. The value is one of the following:

MQOO_INPUT_EXCLUSIVE

Open queue to get messages with exclusive access.

The queue is opened for use with subsequent MQGET calls. The call fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open by this or another application for input of any type (MQOO_INPUT_SHARED or MQOO_INPUT_EXCLUSIVE).

MQOO_INPUT_SHARED

Open queue to get messages with shared access.

The queue is opened for use with subsequent MQGET calls. The call can succeed if the queue is currently open by this or another application with MQOO_INPUT_SHARED, but fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open with MQOO_INPUT_EXCLUSIVE.

To determine the value of this attribute, use the MQIA_DEF_INPUT_OPEN_OPTION selector with the MQINQ call.

DefPersistence (MQLONG)

This is the default persistence of messages on the queue. It applies if MQPER_PERSISTENCE_AS_Q_DEF is specified in the message descriptor when the message is put.

Local	Model	Alias	Remote	Cluster
X	X	X	X	X

If there is more than one definition in the queue-name resolution path, the default persistence is taken from the value of this attribute in the *first* definition in the path at the time of the MQPUT or MQPUT1 call. This could be:

- An alias queue
- A local queue
- A local definition of a remote queue
- A queue-manager alias
- A transmission queue (for example, the *DefXmitQName* queue)

The value is one of the following:

MQPER_PERSISTENT

The message survives system failures and queue manager restarts. Persistent messages cannot be placed on:

- Temporary dynamic queues
- Shared queues that map to a CFSTRUCT object at CFLEVEL(2) or below, or where the CFSTRUCT object is defined as RECOVER(NO).

Persistent messages can be placed on permanent dynamic queues, and predefined queues.

MQPER_NOT_PERSISTENT

The message does not normally survive system failures or queue manager restarts. This applies even if an intact copy of the message is found on auxiliary storage during a queue manager restart.

In the case of shared queues, nonpersistent messages *do* survive restarts of queue managers in the queue-sharing group, but do not survive failures of the coupling facility used to store messages on the shared queues.

Both persistent and nonpersistent messages can exist on the same queue.

To determine the value of this attribute, use the MQIA_DEF_PERSISTENCE selector with the MQINQ call.

DefPriority (MQLONG)

This is the default priority for messages on the queue. This applies if MQPRI_PRIORITY_AS_Q_DEF is specified in the message descriptor when the message is put on the queue.

Local	Model	Alias	Remote	Cluster
X	X	X	X	X

If there is more than one definition in the queue-name resolution path, the default priority for the message is taken from the value of this attribute in the *first* definition in the path at the time of the put operation. This could be:

- An alias queue
- A local queue
- A local definition of a remote queue
- A queue-manager alias
- A transmission queue (for example, the *DefXmitQName* queue)

The way in which a message is placed on a queue depends on the value of the queue's *MsgDeliverySequence* attribute:

- If the *MsgDeliverySequence* attribute is MQMDS_PRIORITY, the logical position at which a message is placed on the queue depends on the value of the *Priority* field in the message descriptor.
- If the *MsgDeliverySequence* attribute is MQMDS_FIFO, messages are placed on the queue as though they had a priority equal to the *DefPriority* of the resolved queue, regardless of the value of the *Priority* field in the message descriptor. However, the *Priority* field retains the value specified by the application that put the message. See *MsgDeliverySequence* attribute for more information.

Priorities are in the range zero (lowest) through *MaxPriority* (highest); see *MaxPriority* attribute.

To determine the value of this attribute, use the MQIA_DEF_PRIORITY selector with the MQINQ call.

DefReadAhead (MQLONG)

Specifies the default read ahead behavior for non-persistent messages delivered to the client.

Local	Model	Alias	Remote	Cluster
X	X	X		

DefReadAhead can be set to one of the following values::

MQREADA_NO

Non-persistent messages are not sent ahead to the client before an application requests them. A maximum of one non-persistent message can be lost if the client ends abnormally.

MQREADA_YES

Non-persistent messages are sent ahead to the client before an application requests them. Non-persistent messages can be lost if the client ends abnormally or if the client does not consume all the messages it is sent.

MQREADA_DISABLED

Read ahead of non-persistent messages is not enabled for this queue. Messages are not sent ahead to the client regardless of whether read ahead is requested by the client application.

To determine the value of this attribute, use the MQIA_DEF_READ_AHEAD selector with the MQINQ call.

DefPResp (MQLONG)

The default put response type (DEFPRESP) attribute defines the value used by applications when the PutResponseType within MQPMO has been set to MQPMO_RESPONSE_AS_Q_DEF. This attribute is valid for all queue types.

Local	Model	Alias	Remote	Cluster
Yes	Yes	Yes	Yes	Yes

The value is one of the following:

SYNC The put operation is issued synchronously returning a response.

ASYNC

The put operation is issued asynchronously, returning a subset of MQMD fields.

To determine the value of this attribute, use the MQIA_DEF_PUT_RESPONSE_TYPE selector with the MQINQ call.

DistLists (MQLONG)

This indicates whether distribution-list messages can be placed on the queue.

Local	Model	Alias	Remote	Cluster
X	X			

A message channel agent (MCA) sets the attribute 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 that next receives the message, after it has been removed from the local transmission queue by a sending MCA.

The sending MCA sets the attribute 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 that the partnering queue manager can process 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 Usage attribute).

The value is one of the following:

MQDL_SUPPORTED

Distribution-list messages can be stored on the queue, and transmitted to the partnering queue manager in that form. This reduces the amount of processing required to send the message to multiple destinations.

MQDL_NOT_SUPPORTED

Distribution-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 are processed correctly by the partnering queue manager.

To determine the value of this attribute, use the MQIA_DIST_LISTS selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

This attribute is not supported on z/OS.

HardenGetBackout (MQLONG)

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.

Local	Model	Alias	Remote	Cluster
X	X			

This count is available in the *BackoutCount* field in the message descriptor after the MQGET call has completed.

The message backout count survives restarts of the queue manager. However, to ensure that the count is accurate, information has to be *hardened* (recorded on disk or other permanent storage device) each time that an MQGET call retrieves a message within a unit of work for this queue. If this is not done, the queue manager fails, and the MQGET call backs out, the count might or might not be incremented.

Hardening information for each MQGET call within a unit of work, however, imposes a performance overhead, so set the *HardenGetBackout* attribute to MQQA_BACKOUT_HARDENED only if it is essential that the count is accurate.

On i5/OS, UNIX systems, and Windows, the message backout count is always hardened, regardless of the setting of this attribute.

The following values are possible:

MQQA_BACKOUT_HARDENED

Hardening is used to ensure that the backout count for messages on this queue is accurate.

MQQA_BACKOUT_NOT_HARDENED

Hardening is not used to ensure that the backout count for messages on this queue is accurate. The count might therefore be lower than it should be.

To determine the value of this attribute, use the MQIA_HARDEN_GET_BACKOUT selector with the MQINQ call.

IndexType (MQLONG)

This specifies the type of index that the queue manager maintains for messages on the queue.

Local	Model	Alias	Remote	Cluster
X	X			

The type of index required depends on how the application retrieves messages, and whether the queue is a shared queue or a nonshared queue (see QSGDisp attribute). The following values are possible for *IndexType*:

MQIT_NONE

No index is maintained by the queue manager for this queue. Use this value for queues that are usually processed sequentially, that is, without using any selection criteria on the MQGET call.

MQIT_MSG_ID

The queue manager maintains an index that uses the message identifiers of the messages on the queue. Use this value for queues where the application usually retrieves messages using the message identifier as the selection criterion on the MQGET call.

MQIT_CORREL_ID

The queue manager maintains an index that uses the correlation identifiers of the messages on the queue. Use this value for queues where the application usually retrieves messages using the correlation identifier as the selection criterion on the MQGET call.

MQIT_MSG_TOKEN

The queue manager maintains an index that uses the message tokens of the messages on the queue for use with the workload manager (WLM) functions of z/OS.

You *must* specify this option for WLM-managed queues; do not specify it for any other type of queue. Also, do not use this value for a queue where an application is not using the z/OS workload manager functions, but is retrieving messages using the message token as a selection criterion on the MQGET call.

MQIT_GROUP_ID

The queue manager maintains an index that uses the group identifiers of the messages on the queue. This value *must* be used for queues where the application retrieves messages using the MQGMO_LOGICAL_ORDER option on the MQGET call.

A queue with this index type cannot be a transmission queue. A shared queue with this index type must be defined to map to a CFSTRUCT object at CFLEVEL(3) or CFLEVEL(4).

Note:

1. The physical order of messages on a queue with index type MQIT_GROUP_ID is not defined, as the queue is optimized for efficient retrieval of messages using the MQGMO_LOGICAL_ORDER option on the MQGET call. This means that the physical order of the messages is not usually the order in which the messages arrived on the queue.
2. If an MQIT_GROUP_ID queue has a *MsgDeliverySequence* of MQMDS_PRIORITY, the queue manager uses message priorities zero and one to optimize the retrieval of messages in logical order. As a result, the first message in a group must not have a priority of zero or one; if it does, the message is processed as though it had a priority of two. The *Priority* field in the MQMD structure is not changed.

For more information about message groups, see the description of the group and segment options in MQGMO - Options field.

The index type that should be used in various cases is shown in Table 97 and Table 98.

Table 97. Recommended or required values of queue index type when MQGMO_LOGICAL_ORDER not specified

Selection criteria on MQGET call	Index type for nonshared queue	Index type for shared queue
None	Any	Any
Selection using one identifier:		
Message identifier	MQIT_MSG_ID recommended	MQIT_NONE or MQIT_MSG_ID required; MQIT_MSG_ID recommended
Correlation identifier	MQIT_CORREL_ID recommended	MQIT_CORREL_ID required
Group identifier	MQIT_GROUP_ID recommended	MQIT_GROUP_ID required
Selection using two identifiers:		
Message identifier plus correlation identifier	MQIT_MSG_ID or MQIT_CORREL_ID recommended	MQIT_MSG_ID or MQIT_CORREL_ID required
Message identifier plus group identifier	MQIT_MSG_ID or MQIT_GROUP_ID recommended	Not supported
Correlation identifier plus group identifier	MQIT_CORREL_ID or MQIT_GROUP_ID recommended	Not supported
Selection using three identifiers:		
Message identifier plus correlation identifier plus group identifier	MQIT_MSG_ID or MQIT_CORREL_ID or MQIT_GROUP_ID recommended	Not supported
Selection using group-related criteria:		
Group identifier plus message sequence number	MQIT_GROUP_ID required	MQIT_GROUP_ID required
Message sequence number (must be 1)	MQIT_GROUP_ID required	MQIT_GROUP_ID required
Selection using message token:		
Message token for application use	Do not use MQIT_MSG_TOKEN	
Message token for WLM use	MQIT_MSG_TOKEN required	Not supported

Table 98. Recommended or required values of queue index type when MQGMO_LOGICAL_ORDER specified

Selection criteria on MQGET call	Index type for nonshared queue	Index type for shared queue
None	MQIT_GROUP_ID required	MQIT_GROUP_ID required
Selection using one identifier:		
Message identifier	MQIT_GROUP_ID required	Not supported

Table 98. Recommended or required values of queue index type when MQGMO_LOGICAL_ORDER specified (continued)

Selection criteria on MQGET call	Index type for nonshared queue	Index type for shared queue
Correlation identifier	MQIT_GROUP_ID required	Not supported
Group identifier	MQIT_GROUP_ID required	MQIT_GROUP_ID required
Selection using two identifiers:		
Message identifier plus correlation identifier	MQIT_GROUP_ID required	Not supported
Message identifier plus group identifier	MQIT_GROUP_ID required	Not supported
Correlation identifier plus group identifier	MQIT_GROUP_ID required	Not supported
Selection using three identifiers:		
Message identifier plus correlation identifier plus group identifier	MQIT_GROUP_ID required	Not supported

To determine the value of this attribute, use the MQIA_INDEX_TYPE selector with the MQINQ call.

This attribute is supported only on z/OS.

InhibitGet (MQLONG)

This controls whether get operations for this queue are allowed.

Local	Model	Alias	Remote	Cluster
X	X	X		

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, for the MQGET call to succeed. The value is one of the following:

MQQA_GET_INHIBITED

Get operations are inhibited.

MQGET calls fail with reason code MQRC_GET_INHIBITED. This includes MQGET calls that specify MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT.

Note: If an MQGET call operating within a unit of work completes successfully, changing the value of the *InhibitGet* attribute subsequently to MQQA_GET_INHIBITED does not prevent the unit of work being committed.

MQQA_GET_ALLOWED

Get operations are allowed.

To determine the value of this attribute, use the MQIA_INHIBIT_GET selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

InhibitPut (MQLONG)

This controls whether put operations for this queue are allowed.

Local	Model	Alias	Remote	Cluster
X	X	X	X	X

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, for the MQPUT or MQPUT1 call to succeed. The value is one of the following:

MQQA_PUT_INHIBITED

Put operations are inhibited.

MQPUT and MQPUT1 calls fail with reason code MQRC_PUT_INHIBITED.

Note: If an MQPUT call operating within a unit of work completes successfully, changing the value of the *InhibitPut* attribute subsequently to MQQA_PUT_INHIBITED does not prevent the unit of work being committed.

MQQA_PUT_ALLOWED

Put operations are allowed.

To determine the value of this attribute, use the MQIA_INHIBIT_PUT selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

InitiationQName (MQCHAR48)

This is the name of a queue defined on the local queue manager; the queue must be of type MQQT_LOCAL.

Local	Model	Alias	Remote	Cluster
X	X			

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 that starts the appropriate application after receipt of the trigger message.

To determine the value of this attribute, use the MQCA_INITIATION_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

MaxMsgLength (MQLONG)

This is an upper limit for the length of the longest *physical* message that can be placed on the queue.

Local	Model	Alias	Remote	Cluster
X	X			

However, because the *MaxMsgLength* queue attribute can be set independently of the *MaxMsgLength* queue-manager attribute, the actual upper limit for the length of the longest physical message that can be placed on the queue is the lesser of those two values.

If the queue manager supports segmentation, it is possible for an application to put a *logical* message that is longer than the lesser of the two *MaxMsgLength* attributes, but only if the application specifies the MQMF_SEGMENTATION_ALLOWED flag in MQMD. If that flag is specified, the upper limit for the length of a logical message is 999 999 999 bytes, but usually resource constraints imposed by the operating system, or by the environment in which the application is running, result in a lower limit.

An attempt to place on the queue a message that is too long fails with one of the following reason codes:

- MQRC_MSG_TOO_BIG_FOR_Q if the message is too big for the queue
- MQRC_MSG_TOO_BIG_FOR_Q_MGR if the message is too big for the queue manager, but not too big for the queue

The lower limit for the *MaxMsgLength* attribute is zero; the upper limit is 100 MB (104 857 600 bytes).

For more information, see MQPUT - BufferLength parameter.

To determine the value of this attribute, use the MQIA_MAX_MSG_LENGTH selector with the MQINQ call.

MaxQDepth (MQLONG)

This is the defined upper limit for the number of physical messages that can exist on the queue at any one time.

Local	Model	Alias	Remote	Cluster
X	X			

An attempt to put a message on a queue that already contains *MaxQDepth* messages fails with reason code MQRC_Q_FULL.

Unit-of-work processing and the segmentation of messages can both cause the actual number of physical messages on the queue to exceed *MaxQDepth*. However, this does not affect the retrievability of the messages; *all* messages on the queue can be retrieved using the MQGET call.

The value of this attribute is zero or greater. The upper limit is determined by the environment:

- On AIX, HP-UX, z/OS, Solaris, Linux, and Windows, the value cannot exceed 999 999 999.
- On i5/OS, the value cannot exceed 640 000.

Note: The storage space available to the queue might be exhausted even if there are fewer than *MaxQDepth* messages on the queue.

To determine the value of this attribute, use the MQIA_MAX_Q_DEPTH selector with the MQINQ call.

MsgDeliverySequence (MQLONG)

Local	Model	Alias	Remote	Cluster
X	X			

This determines the order in which the MQGET call returns messages to the application :

MQMDS_FIFO

Messages are returned in FIFO order (first in, first out).

An MQGET call returns the *first* message that satisfies the selection criteria specified on the call, regardless of the priority of the message.

MQMDS_PRIORITY

Messages are returned in priority order.

An MQGET call returns the *highest-priority* message that satisfies the selection criteria specified on the call. Within each priority level, messages are returned in FIFO order (first in, first out).

- On z/OS, if the queue has an *IndexType* of MQIT_GROUP_ID, the *MsgDeliverySequence* attribute specifies the order in which message groups are returned to the application. The particular sequence in which the groups are returned is determined by the position or priority of the first message in each group. The physical order of messages on the queue is not defined, as the queue is optimized for efficient retrieval of messages using the MQGMO_LOGICAL_ORDER option on the MQGET call.
- On z/OS, if *IndexType* is MQIT_GROUP_ID and *MsgDeliverySequence* is MQMDS_PRIORITY, the queue manager uses message priorities zero and one to optimize the retrieval of messages in logical order. As a result, the first message in a group must not have a priority of zero or one; if it does, the message is processed as though it had a priority of two. The *Priority* field in the MQMD structure is not changed.

If the relevant attributes are changed while there are messages on the queue, the delivery sequence is as follows:

- The order in which messages are returned by the MQGET call is determined by the values of the *MsgDeliverySequence* and *DefPriority* attributes in force for the queue at the time that the message arrives on the queue:
 - If *MsgDeliverySequence* is MQMDS_FIFO when the message arrives, the message is placed on the queue as though its priority were *DefPriority*. This does not affect the value of the *Priority* field in the message descriptor of the message; that field retains the value it had when the message was first put.
 - If *MsgDeliverySequence* is MQMDS_PRIORITY when the message arrives, the message is placed on the queue at the place appropriate to the priority given by the *Priority* field in the message descriptor.

If the value of the *MsgDeliverySequence* attribute is changed while there are messages on the queue, the order of the messages on the queue is not changed.

If the value of the *DefPriority* attribute is changed while there are messages on the queue, the messages are not necessarily delivered in FIFO order, even though the *MsgDeliverySequence* attribute is set to MQMDS_FIFO; those that were placed on the queue at the higher priority are delivered first.

To determine the value of this attribute, use the MQIA_MSG_DELIVERY_SEQUENCE selector with the MQINQ call.

NonPersistentMessageClass (MQLONG)

The reliability goal for nonpersistent messages.

Local	Model	Alias	Remote	Cluster
X	X			

This specifies the circumstances under which nonpersistent messages put on this queue are discarded:

MQNPM_CLASS_NORMAL

Nonpersistent messages are limited to the lifetime of the queue manager session; the messages are discarded in the event of a queue manager restart. This is valid only for non-shared queues, and is the default value.

MQNPM_CLASS_HIGH

The queue manager attempts to retain nonpersistent messages for the lifetime of the queue. Nonpersistent messages might still be lost in the event of a failure. This value is enforced for shared queues.

To determine the value of this attribute, use the MQIA_NPM_CLASS selector with the MQINQ call.

OpenInputCount (MQLONG)

This is the number of handles that are currently valid for removing messages from the queue by means of the MQGET call.

Local	Model	Alias	Remote	Cluster
X				

It is the total number of such handles known to the *local* queue manager. If the queue is a shared queue, the count does not include opens for input that were performed for the queue at other queue managers in the queue-sharing group to which the local queue manager belongs.

The count includes handles where an alias queue that resolves to this queue was opened for input. The count does not include handles where the queue was opened for actions that did not include input (for example, a queue opened only for browse).

The value of this attribute fluctuates as the queue manager operates.

To determine the value of this attribute, use the MQIA_OPEN_INPUT_COUNT selector with the MQINQ call.

OpenOutputCount (MQLONG)

This is the number of handles that are currently valid for adding messages to the queue by means of the MQPUT call.

Local	Model	Alias	Remote	Cluster
X				

It is the total number of such handles known to the *local* queue manager; it does not include opens for output that were performed for this queue at remote queue managers. If the queue is a shared queue, the count does not include opens for output that were performed for the queue at other queue managers in the queue-sharing group to which the local queue manager belongs.

The count includes handles where an alias queue that resolves to this queue was opened for output. The count does not include handles where the queue was opened for actions that did not include output (for example, a queue opened only for inquire).

The value of this attribute fluctuates as the queue manager operates.

To determine the value of this attribute, use the MQIA_OPEN_OUTPUT_COUNT selector with the MQINQ call.

ProcessName (MQCHAR48)

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.

Local	Model	Alias	Remote	Cluster
X	X			

To determine the value of this attribute, use the MQCA_PROCESS_NAME selector with the MQINQ call. The length of this attribute is given by MQ_PROCESS_NAME_LENGTH.

PropertyControl (MQLONG)

Specifies how message properties are handled for messages that are retrieved from queues using the MQGET call with the MQGMO_PROPERTIES_AS_Q_DEF option.

Local	Model	Alias	Remote	Cluster
X	X	X		

The value is one of the following:

MQPROP_ALL

All properties of the message are included with the message when it is delivered to the application. The properties, except those in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

MQPROP_COMPATIBILITY

If the message contains a property with a prefix of mcd., jms., usr. or mqext., all message properties are delivered to the application in an MQRFH2 header. Otherwise all properties of the message, except those contained in the message descriptor (or extension), are discarded and are no longer accessible to the application. This is the default value; it allows applications which expect JMS related properties to be in an MQRFH2 header in the message data to continue to work unmodified.

MQPROP_FORCE_MQRFH2

Properties are always returned in the message data in an MQRFH2 header regardless of whether the application specifies a message handle. A valid

message handle supplied in the `MsgHandle` field of the `MQGMO` structure on the `MQGET` call is ignored. Properties of the message are not accessible via the message handle.

MQPROP_NONE

All properties of the message, except those in the message descriptor (or extension), are removed from the message before the message is delivered to the application.

This parameter is applicable to Local, Alias and Model queues. To determine its value, use the `MQIA_PROPERTY_CONTROL` selector with the `MQINQ` call.

QDepthHighEvent (MQLONG)

This controls whether Queue Depth High events are generated.

Local	Model	Alias	Remote	Cluster
X	X			

A Queue Depth High event indicates that an application has put a message on a queue, and this has caused the number of messages on the queue to become greater than or equal to the queue depth high threshold (see the *QDepthHighLimit* attribute).

Note: The value of this attribute can change dynamically.

The value is one of the following:

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

For more information about events, see *Monitoring WebSphere MQ*.

To determine the value of this attribute, use the `MQIA_Q_DEPTH_HIGH_EVENT` selector with the `MQINQ` call.

This attribute is supported on z/OS, but the `MQINQ` call cannot be used to determine its value.

QDepthHighLimit (MQLONG)

This is the threshold against which the queue depth is compared to generate a Queue Depth High event.

Local	Model	Alias	Remote	Cluster
X	X			

This event indicates that an application has put a message on a queue, and that this has caused the number of messages on the queue to become greater than or equal to the queue depth high threshold. See *QDepthHighEvent* attribute.

The value is expressed as a percentage of the maximum queue depth (*MaxQDepth* attribute), and is greater than or equal to 0 and less than or equal to 100. The default value is 80.

To determine the value of this attribute, use the MQIA_Q_DEPTH_HIGH_LIMIT selector with the MQINQ call.

This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value.

QDepthLowEvent (MQLONG)

This controls whether Queue Depth Low events are generated.

Local	Model	Alias	Remote	Cluster
X	X			

A Queue Depth Low event indicates that an application has retrieved a message from a queue, and that this has caused the number of messages on the queue to become less than or equal to the queue depth low threshold (see QDepthLowLimit attribute).

Note: The value of this attribute can change dynamically.

The value is one of the following:

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

For more information about events, see the *WebSphere MQ Monitoring* book.

To determine the value of this attribute, use the MQIA_Q_DEPTH_LOW_EVENT selector with the MQINQ call.

This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value.

QDepthLowLimit (MQLONG)

This is the threshold against which the queue depth is compared to generate a Queue Depth Low event.

Local	Model	Alias	Remote	Cluster
X	X			

This event indicates that an application has retrieved a message from a queue, and that this has caused the number of messages on the queue to become less than or equal to the queue depth low threshold. See QDepthLowEvent attribute.

The value is expressed as a percentage of the maximum queue depth (*MaxQDepth* attribute), and is greater than or equal to 0 and less than or equal to 100. The default value is 20.

To determine the value of this attribute, use the MQIA_Q_DEPTH_LOW_LIMIT selector with the MQINQ call.

This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value.

QDepthMaxEvent (MQLONG)

This 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.

Local	Model	Alias	Remote	Cluster
X	X			

Note: The value of this attribute can change dynamically.

The value is one of the following:

MQEVR_DISABLED

Event reporting disabled.

MQEVR_ENABLED

Event reporting enabled.

For more information about events, see *WebSphere MQ Monitoring*.

To determine the value of this attribute, use the MQIA_Q_DEPTH_MAX_EVENT selector with the MQINQ call.

This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value.

QDesc (MQCHAR64)

Use this field for descriptive commentary.

Local	Model	Alias	Remote	Cluster
X	X	X	X	X

The content of the field is of no significance to the queue manager, but the queue manager might 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 might be translated incorrectly if this field is sent to another queue manager.

To determine the value of this attribute, use the MQCA_Q_DESC selector with the MQINQ call. The length of this attribute is given by MQ_Q_DESC_LENGTH.

QName (MQCHAR48)

This is the name of a queue defined on the local queue manager.

Local	Model	Alias	Remote	Cluster
X		X	X	X

For more information about queue names, see the *WebSphere MQ Application Programming Guide*. All queues defined on a queue manager share the same queue name space. Therefore, an MQQT_LOCAL queue and an MQQT_ALIAS queue cannot have the same name.

To determine the value of this attribute, use the MQCA_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

QServiceInterval (MQLONG)

This is the service interval used for comparison to generate Service Interval High and Service Interval OK events.

Local	Model	Alias	Remote	Cluster
X	X			

See QServiceIntervalEvent attribute.

The value is in units of milliseconds, and is greater than or equal to zero, and less than or equal to 999 999 999.

To determine the value of this attribute, use the MQIA_Q_SERVICE_INTERVAL selector with the MQINQ call.

This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value.

QServiceIntervalEvent (MQLONG)

This controls whether Service Interval High or Service Interval OK events are generated.

Local	Model	Alias	Remote	Cluster
X	X			

- A Service Interval High event is generated when a check indicates that no messages have been retrieved from the queue for at least the time indicated by the *QServiceInterval* attribute.
- A Service Interval OK event is generated when a check indicates that messages have been retrieved from the queue within the time indicated by the *QServiceInterval* attribute.

Note: The value of this attribute can change dynamically.

The value is one of the following:

MQQSIE_HIGH

Queue Service Interval High events enabled.

- Queue Service Interval High events are **enabled** and
- Queue Service Interval OK events are **disabled**.

MQQSIE_OK

Queue Service Interval OK events enabled.

- Queue Service Interval High events are **disabled** and
- Queue Service Interval OK events are **enabled**.

MQQSIE_NONE

No queue service interval events enabled.

- Queue Service Interval High events are **disabled** and
- Queue Service Interval OK events are also **disabled**.

For shared queues, the value of this attribute is ignored; the value MQQSIE_NONE is assumed.

For more information about events, see *WebSphere MQ Monitoring*.

To determine the value of this attribute, use the MQIA_Q_SERVICE_INTERVAL_EVENT selector with the MQINQ call.

On z/OS, you cannot use the MQINQ call to determine the value of this attribute.

QSGDisp (MQLONG)

This specifies the disposition of the queue.

Local	Model	Alias	Remote	Cluster
X		X	X	

The value is one of the following:

MQQSGD_Q_MGR

The object has queue-manager disposition. This means that the object definition is known only to the local queue manager; the definition is not known to other queue managers in the queue-sharing group.

Each queue manager in the queue-sharing group can have an object with the same name and type as the current object, but these are separate objects and there is no correlation between them. Their attributes are not constrained to be the same as each other.

MQQSGD_COPY

The object is a local copy of a master object definition that exists in the shared repository. Each queue manager in the queue-sharing group can have its own copy of the object. Initially, all copies have the same attributes, but by using MQSC commands, you can alter each copy so that its attributes differ from those of the other copies. The attributes of the copies are resynchronized when the master definition in the shared repository is altered.

MQQSGD_SHARED

The object has shared disposition. This means that there exists in the shared repository a single instance of the object that is known to all queue managers in the queue-sharing group. When a queue manager in the group accesses the object, it accesses the single shared instance of the object.

To determine the value of this attribute, use the MQIA_QSG_DISP selector with the MQINQ call.

This attribute is supported only on z/OS.

QueueAccounting (MQCHAR12)

Local	Model	Alias	Remote	Cluster
X	X	X	X	

This controls the collection of accounting data for the queue. For accounting data to be collected for this queue, accounting data for this connection must also be enabled, using either the QMGR attribute ACCTQ or the Options field in the MQCNO structure on the MQCONN call.

This attribute has one of the following values:

MQMON_Q_MGR

Accounting data for this queue is collected based on the setting of the QMGR attribute ACCTQ. This is the default setting.

MQMON_OFF

Do not collect accounting data for this queue.

MQMON_ON

Collect accounting data for this queue.

QueueMonitoring (MQLONG)

Controls the collection of online monitoring data for queues.

Local	Model	Alias	Remote	Cluster
X	X			

The value is one of the following:

MQMON_QMGR

Collect monitoring data according to the setting of the *QueueMonitoring* queue manager attribute. This is the default value.

MQMON_OFF

Online monitoring data collection is turned off for this queue.

MQMON_LOW

If the value of the *QueueMonitoring* queue manager attribute is not MQMON_NONE, online monitoring data collection is turned on, with a low rate of data collection for this queue.

MQMON_MEDIUM

If the value of the *QueueMonitoring* queue manager attribute is not MQMON_NONE, online monitoring data collection is turned on, with a moderate rate of data collection for this queue.

MQMON_HIGH

If the value of the *QueueMonitoring* queue manager attribute is not MQMON_NONE, online monitoring data collection is turned on, with a high rate of data collection for this queue.

To determine the value of this attribute, use the MQIA_MONITORING_Q selector with the MQINQ call.

QueueStatistics (MQCHAR12)

Local	Model	Alias	Remote	Cluster
X	X	X	X	

This controls the collection of statistics data for the queue.

This attribute has one of the following values:

MQMON_Q_MGR

Accounting data for this queue is collected based on the setting of the QMGR attribute STATQ. This is the default setting.

MQMON_OFF

Switch off statistics data collection for this queue.

MQMON_ON

Switch on statistics data collection for this queue.

QType (MQLONG)

Local	Model	Alias	Remote	Cluster
X		X	X	X

This is the type of queue; it has one of the following values:

MQQT_ALIAS

Alias queue definition.

MQQT_CLUSTER

Cluster queue.

MQQT_LOCAL

Local queue.

MQQT_REMOTE

Local definition of a remote queue.

To determine the value of this attribute, use the MQIA_Q_TYPE selector with the MQINQ call.

RemoteQMgrName (MQCHAR48)

Local	Model	Alias	Remote	Cluster
			X	

This is the name of the remote queue manager on which the queue *RemoteQName* is defined. If the *RemoteQName* queue has a *QSGDisp* value of MQQSGD_COPY or MQQSGD_SHARED, *RemoteQMgrName* can be the name of the queue-sharing group that owns *RemoteQName*.

If an application opens the local definition of a remote queue, *RemoteQMgrName* must not be blank and must not be the name of the local queue manager. If *XmitQName* is blank, the local queue whose name is the same as *RemoteQMgrName* is

used as the transmission queue. If there is no queue with the name *RemoteQMgrName*, the queue identified by the *DefXmitQName* queue-manager attribute is used.

If this definition is used for a queue-manager alias, *RemoteQMgrName* is the name of the queue manager that is being aliased. It can be the name of the local queue manager. Otherwise, if *XmitQName* is blank when the open occurs, there must be a local queue whose name is the same as *RemoteQMgrName*; this queue is used as the transmission queue.

If this definition is used for a reply-to alias, this name is the name of the queue manager that is to be the *ReplyToQMgr*.

Note: No validation is performed on the value specified for this attribute when the queue definition is created or modified.

To determine the value of this attribute, use the MQCA_REMOTE_Q_MGR_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_MGR_NAME_LENGTH.

RemoteQName (MQCHAR48)

Local	Model	Alias	Remote	Cluster
			X	

This is the name of the queue as it is known on the remote queue manager *RemoteQMgrName*.

If an application opens the local definition of a remote queue, when the open occurs *RemoteQName* must not be blank.

If this definition is used for a queue-manager alias definition, when the open occurs *RemoteQName* must be blank.

If the definition is used for a reply-to alias, this name is the name of the queue that is to be the *ReplyToQ*.

Note: No validation is performed on the value specified for this attribute when the queue definition is created or modified.

To determine the value of this attribute, use the MQCA_REMOTE_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

RetentionInterval (MQLONG)

This is the period of time for which to retain the queue. After this time has elapsed, the queue is eligible for deletion.

Local	Model	Alias	Remote	Cluster
X	X			

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.

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 take any required action.

Use a realistic retention interval to prevent the accumulation of permanent dynamic queues (see *DefinitionType* attribute). However, this attribute can also be used with predefined queues.

To determine the value of this attribute, use the *MQIA_RETENTION_INTERVAL* selector with the *MQINQ* call.

Scope (MQLONG)

This controls whether an entry for this queue also exists in a cell directory.

Local	Model	Alias	Remote	Cluster
X		X	X	

A cell directory is provided by an installable Name service. The value is one of the following:

MQSCO_Q_MGR

The queue definition has queue-manager scope: the definition of the queue does not extend beyond the queue manager that owns it. To open the queue for output from some other queue manager, either the name of the owning queue manager must be specified, or the other queue manager must have a local definition of the queue.

MQSCO_CELL

The queue definition has cell scope: the queue definition is also placed in a cell directory available to all the queue managers in the cell. The queue can be opened for output from any of the queue managers in the cell by specifying the name of the queue; the name of the queue manager that owns the queue need not be specified. However, the queue definition is not available to any queue manager in the cell that also has a local definition of a queue with that name, as the local definition takes precedence.

A cell directory is provided by an installable Name service.

Model and dynamic queues cannot have cell scope.

This value is only valid if a name service supporting a cell directory has been configured.

To determine the value of this attribute, use the *MQIA_SCOPE* selector with the *MQINQ* call.

Support for this attribute is subject to the following restrictions:

- On i5/OS, the attribute is supported, but only *MQSCO_Q_MGR* is valid.

- On z/OS, the attribute is not supported.

Shareability (MQLONG)

This indicates whether the queue can be opened for input multiple times concurrently.

Local	Model	Alias	Remote	Cluster
X	X			

The value is one of the following:

MQQA_SHAREABLE

Queue is shareable.

Multiple opens with the MQOO_INPUT_SHARED option are allowed.

MQQA_NOT_SHAREABLE

Queue is not shareable.

An MQOPEN call with the MQOO_INPUT_SHARED option is treated as MQOO_INPUT_EXCLUSIVE.

To determine the value of this attribute, use the MQIA_SHAREABILITY selector with the MQINQ call.

StorageClass (MQCHAR8)

This is a user-defined name that defines the physical storage used to hold the queue. In practice, a message is written to disk only if it needs to be paged out of its memory buffer.

Local	Model	Alias	Remote	Cluster
X	X			

To determine the value of this attribute, use the MQCA_STORAGE_CLASS selector with the MQINQ call. The length of this attribute is given by MQ_STORAGE_CLASS_LENGTH.

This attribute is supported only on z/OS.

TriggerControl (MQLONG)

This controls whether trigger messages are written to an initiation queue to start an application to service the queue.

Local	Model	Alias	Remote	Cluster
X	X			

This is one of the following:

MQTC_OFF

No trigger messages are to be written for this queue. The value of *TriggerType* is irrelevant in this case.

MQTC_ON

Trigger messages are to be written for this queue when the appropriate trigger events occur.

To determine the value of this attribute, use the MQIA_TRIGGER_CONTROL selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

TriggerData (MQCHAR64)

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.

Local	Model	Alias	Remote	Cluster
X	X			

The content of this data is of no significance to the queue manager. It is meaningful either to the trigger-monitor application that processes the initiation queue, or to the application that the trigger monitor starts.

The character string must not contain any nulls. It is padded to the right with blanks if necessary.

To determine the value of this attribute, use the MQCA_TRIGGER_DATA selector with the MQINQ call. To change the value of this attribute, use the MQSET call. The length of this attribute is given by MQ_TRIGGER_DATA_LENGTH.

TriggerDepth (MQLONG)

Local	Model	Alias	Remote	Cluster
X	X			

This is the number of messages of priority *TriggerMsgPriority* or greater that must be on the queue before a trigger message is written. This applies when *TriggerType* is set to MQTT_DEPTH. The value of *TriggerDepth* is one or greater. This attribute is not used otherwise.

To determine the value of this attribute, use the MQIA_TRIGGER_DEPTH selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

TriggerMsgPriority (MQLONG)

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 to generate a trigger message).

Local	Model	Alias	Remote	Cluster
X	X			

TriggerMsgPriority can be in the range zero (lowest) through *MaxPriority* (highest; see *MaxPriority* attribute); a value of zero causes all messages to contribute to the generation of trigger messages.

To determine the value of this attribute, use the MQIA_TRIGGER_MSG_PRIORITY selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

TriggerType (MQLONG)

This controls the conditions under which trigger messages are written as a result of messages arriving on this queue.

Local	Model	Alias	Remote	Cluster
X	X			

The value is one of the following:

MQTT_NONE

No trigger messages are written as a result of messages on this queue. This has the same effect as setting *TriggerControl* to MQTC_OFF.

MQTT_FIRST

A trigger message is written whenever the number of messages of priority *TriggerMsgPriority* or greater on the queue changes from 0 to 1.

MQTT EVERY

A trigger message is written whenever a message of priority *TriggerMsgPriority* or greater arrives on the queue.

MQTT_DEPTH

A trigger message is written whenever the number of messages of priority *TriggerMsgPriority* or greater on the queue equals or exceeds *TriggerDepth*. After the trigger message has been written, *TriggerControl* is set to MQTC_OFF to prevent further triggering until it is explicitly turned on again.

To determine the value of this attribute, use the MQIA_TRIGGER_TYPE selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

Usage (MQLONG)

This indicates what the queue is used for.

Local	Model	Alias	Remote	Cluster
X	X			

The value is one of the following:

MQUS_NORMAL

This is a queue that applications use when putting and getting messages; the queue is not a transmission queue.

MQUS_TRANSMISSION

This is a queue used to hold messages destined for remote queue managers. When an application sends a message to a remote queue, the local queue manager stores the message temporarily on the appropriate transmission queue in a special format. A message channel agent then reads the message from the transmission queue, and transports the message to the remote queue manager. For more information about transmission queues, see the *WebSphere MQ Application Programming Guide*.

Only privileged applications can open a transmission queue for MQOO_OUTPUT to put messages on it directly. Usually, only utility applications do this. Ensure that the message data format is correct (see “MQXQH – Transmission-queue header” on page 387) or errors might occur during the transmission process. Context is not passed or set unless one of the MQPMO_*_CONTEXT context options is specified.

To determine the value of this attribute, use the MQIA_USAGE selector with the MQINQ call.

XmitQName (MQCHAR48)

This is the 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.

Local	Model	Alias	Remote	Cluster
			X	

If *XmitQName* is blank, the local queue whose name is the same as *RemoteQMGrName* is used as the transmission queue. If there is no queue with the name *RemoteQMGrName*, the queue identified by the *DefXmitQName* queue-manager attribute is used.

This attribute is ignored if the definition is being used as a queue-manager alias and *RemoteQMGrName* is the name of the local queue manager. It is also ignored if the definition is used as a reply-to queue alias definition.

To determine the value of this attribute, use the MQCA_XMIT_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

Attributes for namelists

The following table summarizes the attributes that are specific to namelists. The attributes are described in alphabetic order.

Namelists are supported on all WebSphere MQ systems, plus WebSphere MQ clients connected to these systems.

Note: The names of the attributes shown in this book are descriptive names used with the MQINQ and MQSET calls; the names are the same as for the PCF commands. When MQSC commands are used to define, alter, or display attributes, alternative short names are used; see the *WebSphere MQ Script (MQSC) Command Reference* for details.

Table 99. Attributes for namelists

Attribute	Description	Topic
<i>AlterationDate</i>	Date when definition was last changed	AlterationDate
<i>AlterationTime</i>	Time when definition was last changed	AlterationTime
<i>NameCount</i>	Number of names in namelist	NameCount
<i>NamelistDesc</i>	Namelist description	NamelistDesc
<i>NamelistName</i>	Namelist name	NamelistName
<i>Names</i>	A list of <i>NameCount</i> names	Names

Table 99. Attributes for namelists (continued)

Attribute	Description	Topic
<i>NamelistType</i>	Namelist type	NamelistType
<i>QSGDisp</i>	Queue-sharing group disposition	QSGDisp

AlterationDate (MQCHAR12)

This is the date when the definition was last changed. The format of the date is YYYY-MM-DD, padded with two trailing blanks to make the length 12 bytes.

To determine the value of this attribute, use the MQCA_ALTERATION_DATE selector with the MQINQ call. The length of this attribute is given by MQ_DATE_LENGTH.

AlterationTime (MQCHAR8)

This is the time when the definition was last changed. The format of the time is HH.MM.SS.

To determine the value of this attribute, use the MQCA_ALTERATION_TIME selector with the MQINQ call. The length of this attribute is given by MQ_TIME_LENGTH.

NameCount (MQLONG)

This is the number of names in the namelist. It is greater than or equal to zero. The following value is defined:

MQNC_MAX_NAMELIST_NAME_COUNT
Maximum number of names in a namelist.

To determine the value of this attribute, use the MQIA_NAME_COUNT selector with the MQINQ call.

NamelistDesc (MQCHAR64)

Use this field for descriptive commentary; its value is established by the definition process. The content of the field is of no significance to the queue manager, but the queue manager might 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 might be translated incorrectly if this field is sent to another queue manager.

To determine the value of this attribute, use the MQCA_NAMELIST_DESC selector with the MQINQ call.

The length of this attribute is given by MQ_NAMELIST_DESC_LENGTH.

NamelistName (MQCHAR48)

This is the name of a namelist that is defined on the local queue manager. For more information about namelist names, see the *WebSphere MQ Application Programming Guide*.

Each namelist has a name that is different from the names of other namelists belonging to the queue manager, but might duplicate the names of other queue manager objects of different types (for example, queues).

To determine the value of this attribute, use the MQCA_NAMELIST_NAME selector with the MQINQ call.

The length of this attribute is given by MQ_NAMELIST_NAME_LENGTH.

NamelistType (MQLONG)

This specifies the nature of the names in the namelist, and indicates how the namelist is used. The value is one of the following:

MQNT_NONE

Namelist with no assigned type.

MQNT_Q

Namelist containing the names of queues.

MQNT_CLUSTER

Namelist containing the names of clusters.

MQNT_AUTH_INFO

Namelist containing the names of authentication-information objects.

To determine the value of this attribute, use the MQIA_NAMELIST_TYPE selector with the MQINQ call.

This attribute is supported only on z/OS.

Names (MQCHAR48xNameCount)

This is a list of *NameCount* names, where each name is the name of an object that is defined to the local queue manager. For more information about object names, see the *WebSphere MQ Application Programming Guide*.

To determine the value of this attribute, use the MQCA_NAMES selector with the MQINQ call.

The length of each name in the list is given by MQ_OBJECT_NAME_LENGTH.

QSGDisp (MQLONG)

This specifies the disposition of the namelist. The value is one of the following:

MQQSGD_Q_MGR

The object has queue-manager disposition: the object definition is known only to the local queue manager; the definition is not known to other queue managers in the queue-sharing group.

Each queue manager in the queue-sharing group can have an object with the same name and type as the current object, but these are separate objects and there is no correlation between them. Their attributes are not constrained to be the same as each other.

MQQSGD_COPY

The object is a local copy of a master object definition that exists in the shared repository. Each queue manager in the queue-sharing group can have its own copy of the object. Initially, all copies have the same attributes, but you can alter each copy, using MQSC commands, so that its attributes differ from those of the other copies. The attributes of the copies are resynchronized when the master definition in the shared repository is altered.

To determine the value of this attribute, use the MQIA_QSG_DISP selector with the MQINQ call.

This attribute is supported only on z/OS.

Attributes for process definitions

The following table summarizes the attributes that are specific to process definitions. The attributes are described in alphabetic order.

Note: The names of the attributes shown in this book are descriptive names used with the MQINQ and MQSET calls; the names are the same as for the PCF commands. When MQSC commands are used to define, alter, or display attributes, alternative short names are used; see the *WebSphere MQ Script (MQSC) Command Reference* for details.

Table 100. Attributes for process definitions

Attribute	Description	Topic
<i>AlterationDate</i>	Date when definition was last changed	AlterationDate
<i>AlterationTime</i>	Time when definition was last changed	AlterationTime
<i>AppId</i>	Application identifier	AppId
<i>AppType</i>	Application type	AppType
<i>EnvData</i>	Environment data	EnvData
<i>ProcessDesc</i>	Process description	ProcessDesc
<i>ProcessName</i>	Process name	ProcessName
<i>QSGDisp</i>	Queue-sharing group disposition	QSGDisp
<i>UserData</i>	User data	UserData

AlterationDate (MQCHAR12)

This is the date when the definition was last changed. The format of the date is YYYY-MM-DD, padded with two trailing blanks to make the length 12 bytes.

To determine the value of this attribute, use the MQCA_ALTERATION_DATE selector with the MQINQ call. The length of this attribute is given by MQ_DATE_LENGTH.

AlterationTime (MQCHAR8)

This is the time when the definition was last changed. The format of the time is HH.MM.SS.

To determine the value of this attribute, use the MQCA_ALTERATION_TIME selector with the MQINQ call. The length of this attribute is given by MQ_TIME_LENGTH.

ApplId (MQCHAR256)

This is a character string that identifies the application to be started. This information is for use by a trigger-monitor application that processes messages on the initiation queue; the information is sent to the initiation queue as part of the trigger message.

The meaning of *ApplId* is determined by the trigger-monitor application. The trigger monitor provided by WebSphere MQ requires *ApplId* to be the name of an executable program. The following notes apply to the environments indicated:

- On z/OS, *ApplId* must be:
 - A CICS transaction identifier, for applications started using the CICS trigger-monitor transaction CKTI
 - An IMS transaction identifier, for applications started using the IMS trigger monitor CSQQTRMN
- On Windows systems, the program name can be prefixed with a drive and directory path.
- On UNIX systems, the program name can be prefixed with a directory path.

The character string cannot contain any nulls. It is padded to the right with blanks if necessary.

To determine the value of this attribute, use the MQCA_APPL_ID selector with the MQINQ call. The length of this attribute is given by MQ_PROCESS_APPL_ID_LENGTH.

ApplType (MQLONG)

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; restrict user-defined application types to values in the range MQAT_USER_FIRST through MQAT_USER_LAST:

MQAT_AIX

AIX application (same value as MQAT_UNIX).

MQAT_BATCH

Batch application

MQAT_BROKER

Broker application

MQAT_CICS

CICS transaction.

MQAT_CICS_BRIDGE

CICS bridge application.

MQAT_CICS_VSE
CICS/VSE transaction.

MQAT_DOS
WebSphere MQ client application on PC DOS.

MQAT_IMS
IMS application.

| **MQAT_IMS_BRIDGE**
| IMS bridge application.

| **MQAT_JAVA**
| Java application.

MQAT_MVS
MVS or TSO application (same value as MQAT_ZOS).

MQAT_NOTES_AGENT
Lotus Notes Agent application.

MQAT_NSK
Compaq NonStop Kernel application.

MQAT_OS2
OS/2 or Presentation Manager application.

MQAT_OS390
OS/390 application (same value as MQAT_ZOS).

MQAT_OS400
i5/OS application.

| **MQAT_RRS_BATCH**
| RRS batch application.

MQAT_UNIX
UNIX application.

| **MQAT_UNKNOWN**
| Application of unknown type.

| **MQAT_USER**
| User application.

MQAT_VMS
Digital OpenVMS application.

| **MQAT_VOS**
| Stratus VOS application.

MQAT_WINDOWS
16-bit Windows application.

MQAT_WINDOWS_NT
32-bit Windows application.

MQAT_WLM
z/OS workload manager application.

| **MQAT_XCF**
| XCF.

MQAT_ZOS
z/OS application.

MQAT_USER_FIRST

Lowest value for user-defined application type.

MQAT_USER_LAST

Highest value for user-defined application type.

To determine the value of this attribute, use the MQIA_APPL_TYPE selector with the MQINQ call.

EnvData (MQCHAR128)

This is a character string that contains environment-related information pertaining to the application to be started. This information is for use by a trigger-monitor application that processes messages on the initiation queue; the information is sent to the initiation queue as part of the trigger message.

The meaning of *EnvData* is determined by the trigger-monitor application. The trigger monitor provided by WebSphere MQ appends *EnvData* to the parameter list passed to the started application. The parameter list consists of the MQTMC2 structure, followed by one blank, followed by *EnvData* with trailing blanks removed. The following notes apply to the environments indicated:

- On z/OS:
 - *EnvData* is not used by the trigger-monitor applications provided by WebSphere MQ.
 - If ApplType is MQAT_WLM, you can supply default values in *EnvData* for the ServiceName and ServiceStep fields in the work information header (MQWIH).
- On UNIX systems, *EnvData* can be set to the & character to run the started application in the background.

The character string cannot contain any nulls. It is padded to the right with blanks if necessary.

To determine the value of this attribute, use the MQCA_ENV_DATA selector with the MQINQ call. The length of this attribute is given by MQ_PROCESS_ENV_DATA_LENGTH.

ProcessDesc (MQCHAR64)

Use this field for descriptive commentary. The content of the field is of no significance to the queue manager, but the queue manager might 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 might be translated incorrectly if this field is sent to another queue manager.

To determine the value of this attribute, use the MQCA_PROCESS_DESC selector with the MQINQ call.

The length of this attribute is given by MQ_PROCESS_DESC_LENGTH.

ProcessName (MQCHAR48)

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 might be the same as the names of other queue manager objects of different types (for example, queues).

To determine the value of this attribute, use the MQCA_PROCESS_NAME selector with the MQINQ call.

The length of this attribute is given by MQ_PROCESS_NAME_LENGTH.

QSGDisp (MQLONG)

This specifies the disposition of the process definition. The value is one of the following:

MQQSGD_Q_MGR

The object has queue-manager disposition: the object definition is known only to the local queue manager; the definition is not known to other queue managers in the queue-sharing group.

Each queue manager in the queue-sharing group can have an object with the same name and type as the current object, but these are separate objects and there is no correlation between them. Their attributes are not constrained to be the same as each other.

MQQSGD_COPY

The object is a local copy of a master object definition that exists in the shared repository. Each queue manager in the queue-sharing group can have its own copy of the object. Initially, all copies have the same attributes, but you can alter each copy, using MQSC commands, so that its attributes differ from those of the other copies. The attributes of the copies are resynchronized when the master definition in the shared repository is altered.

To determine the value of this attribute, use the MQIA_QSG_DISP selector with the MQINQ call.

This attribute is supported only on z/OS.

UserData (MQCHAR128)

This is a character string that contains user information pertaining to the application to be started. This information is for use by a trigger-monitor application that processes messages on the initiation queue, or the application that is started by the trigger monitor. The information is sent to the initiation queue as part of the trigger message.

The meaning of *UserData* is determined by the trigger-monitor application. The trigger monitor provided by WebSphere MQ passes *UserData* to the started application as part of the parameter list. The parameter list consists of the MQTMC2 structure (containing *UserData*), followed by one blank, followed by *EnvData* with trailing blanks removed.

The character string cannot contain any nulls. It is padded to the right with blanks if necessary.

To determine the value of this attribute, use the MQCA_USER_DATA selector with the MQINQ call. The length of this attribute is given by MQ_PROCESS_USER_DATA_LENGTH.

Chapter 4. Return codes

For each WebSphere MQ Message Queue Interface (MQI) and WebSphere MQ Administration Interface (MQAI) 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.

Applications checking for successful completion following a WebSphere MQ API call should always check the completion code. Do not assume the completion code value, based on the value of the reason code.

Completion codes

The completion code parameter (*CompCode*) allows the caller to see quickly whether the call completed successfully, completed partially, or failed.

The following is a list of completion codes, with more detail than is given in the call descriptions:

MQCC_OK

The call completed fully; all output parameters have been set. The *Reason* parameter always has the value MQRC_NONE in this case.

MQCC_WARNING

The call completed partially. Some output parameters might have been set in addition to the *CompCode* and *Reason* output parameters. The *Reason* parameter gives additional information about the partial completion.

MQCC_FAILED

The processing of the call did not complete. The state of the queue manager is unchanged, except where specifically noted. The *CompCode* and *Reason* output parameters have been set; other parameters are unchanged, except where noted.

The reason might be a fault in the application program, or it might be the result of some situation external to the program, for example the user's authority might have been revoked. The *Reason* parameter gives additional information about the error.

Reason codes

The reason code parameter (*Reason*) qualifies the completion code parameter (*CompCode*).

If there is no special reason to report, MQRC_NONE is returned. A successful call returns MQCC_OK and MQRC_NONE.

If the completion code is either MQCC_WARNING or MQCC_FAILED, the queue manager always reports a qualifying reason; details are given under each call description.

Where user exit routines set completion codes and reasons, they must adhere to these rules. In addition, any special reason values defined by user exits must be less than zero, to ensure that they do not conflict with values defined by the queue manager. Exits can set reasons already defined by the queue manager, where these are appropriate.

Reason codes also occur in:

- The *Reason* field of the MQDLH structure
- The *Feedback* field of the MQMD structure

For complete descriptions of reason codes see:

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

Chapter 5. Rules for validating MQI options

This appendix lists the situations that produce an MQRC_OPTIONS_ERROR reason code from an MQCONN, MQOPEN, MQPUT, MQPUT1, MQGET, or MQCLOSE call.

MQOPEN call

For the options of the MQOPEN call:

- At least *one* of the following must be specified:
 - MQOO_BROWSE
 - MQOO_INPUT_AS_Q_DEF
 - MQOO_INPUT_EXCLUSIVE
 - MQOO_INPUT_SHARED
 - MQOO_INQUIRE
 - MQOO_OUTPUT
 - MQOO_SET
- Only *one* of the following is allowed:
 - MQOO_INPUT_AS_Q_DEF
 - MQOO_INPUT_EXCLUSIVE
 - MQOO_INPUT_SHARED
- Only *one* of the following is allowed:
 - MQOO_BIND_ON_OPEN
 - MQOO_BIND_NOT_FIXED
 - MQOO_BIND_AS_Q_DEF

Note: The options listed above are mutually exclusive. However, as the value of MQOO_BIND_AS_Q_DEF is zero, specifying it with either of the other two bind options does not result in reason code MQRC_OPTIONS_ERROR. MQOO_BIND_AS_Q_DEF is provided to aid program documentation.

- If MQOO_SAVE_ALL_CONTEXT is specified, one of the MQOO_INPUT_* options must also be specified.
- If one of the MQOO_SET_*_CONTEXT or MQOO_PASS_*_CONTEXT options is specified, MQOO_OUTPUT must also be specified.

MQPUT call

For the put-message options:

- The combination of MQPMO_SYNCPOINT and MQPMO_NO_SYNCPOINT is not allowed.
- Only *one* of the following is allowed:
 - MQPMO_DEFAULT_CONTEXT
 - MQPMO_NO_CONTEXT
 - MQPMO_PASS_ALL_CONTEXT
 - MQPMO_PASS_IDENTITY_CONTEXT

- MQPMO_SET_ALL_CONTEXT
- MQPMO_SET_IDENTITY_CONTEXT
- MQPMO_ALTERNATE_USER_AUTHORITY is not allowed (it is valid only on the MQPUT1 call).

MQPUT1 call

For the put-message options, the rules are the same as for the MQPUT call, except for the following:

- MQPMO_ALTERNATE_USER_AUTHORITY is allowed.
- MQPMO_LOGICAL_ORDER is *not* allowed.

MQGET call

For the get-message options:

- Only *one* of the following is allowed:
 - MQGMO_NO_SYNCPOINT
 - MQGMO_SYNCPOINT
 - MQGMO_SYNCPOINT_IF_PERSISTENT
- Only *one* of the following is allowed:
 - MQGMO_BROWSE_FIRST
 - MQGMO_BROWSE_MSG_UNDER_CURSOR
 - MQGMO_BROWSE_NEXT
 - MQGMO_MSG_UNDER_CURSOR
- MQGMO_SYNCPOINT is not allowed with any of the following:
 - MQGMO_BROWSE_FIRST
 - MQGMO_BROWSE_MSG_UNDER_CURSOR
 - MQGMO_BROWSE_NEXT
 - MQGMO_LOCK
 - MQGMO_UNLOCK
- MQGMO_SYNCPOINT_IF_PERSISTENT is not allowed with any of the following:
 - MQGMO_BROWSE_FIRST
 - MQGMO_BROWSE_MSG_UNDER_CURSOR
 - MQGMO_BROWSE_NEXT
 - MQGMO_COMPLETE_MSG
 - MQGMO_UNLOCK
- MQGMO_MARK_SKIP_BACKOUT requires MQGMO_SYNCPOINT to be specified.
- The combination of MQGMO_WAIT and MQGMO_SET_SIGNAL is not allowed.
- If MQGMO_LOCK is specified, one of the following must also be specified:
 - MQGMO_BROWSE_FIRST
 - MQGMO_BROWSE_MSG_UNDER_CURSOR
 - MQGMO_BROWSE_NEXT
- If MQGMO_UNLOCK is specified, only the following are allowed:
 - MQGMO_NO_SYNCPOINT

- MQGMO_NO_WAIT

MQCLOSE call

For the options of the MQCLOSE call:

- The combination of MQCO_DELETE and MQCO_DELETE_PURGE is not allowed.
- Only one of the following is allowed:
 - MQCO_KEEP_SUB
 - MQCO_REMOVE_SUB

MQSUB call

For the options of the MQSUB call:

- At least one of the following must be specified:
 - MQSO_ALTER
 - MQSO_RESUME
 - MQSO_CREATE
- Only one of the following is allowed:
 - MQSO_DURABLE
 - MQSO_NON_DURABLE

Note: The options listed above are mutually exclusive. However, as the value of MQSO_NON_DURABLE is zero, specifying it with MQSO_DURABLE does not result in reason code MQRC_OPTIONS_ERROR. MQSO_NON_DURABLE is provided to aid program documentation.

- The combination of MQSO_GROUP_SUB and MQSO_MANAGED is not allowed.
- MQSO_GROUP_SUB requires MQSO_SET_CORREL_ID to be specified.
- Only one of the following is allowed: MQSO_ANY_USERID
MQSO_FIXED_USERID
- MQSO_NEW_PUBLICATIONS_ONLY is only allowed in combination with MQSO_CREATE.
- The combination of MQSO_PUBLICATIONS_ON_REQUEST and SubLevel greater than 1 is not allowed.
- Only one of the following is allowed:
 - MQSO_WILDCARD_CHAR
 - MQSO_WILDCARD_TOPIC

Chapter 6. Machine encodings

This appendix describes the structure of the *Encoding* field in the message descriptor (see “MQMD – Message descriptor” on page 181).

The *Encoding* field is a 32-bit integer that is divided into four separate subfields; these subfields identify:

- The encoding used for binary integers
- The encoding used for packed-decimal integers
- The encoding used for floating-point numbers
- Reserved bits

Each subfield is identified by a bit mask that has 1-bits in the positions corresponding to the subfield, and 0-bits elsewhere. The bits are numbered such that bit 0 is the most significant bit, and bit 31 the least significant bit. The following masks are defined:

MQENC_INTEGER_MASK

Mask for binary-integer encoding.

This subfield occupies bit positions 28 through 31 within the *Encoding* field.

MQENC_DECIMAL_MASK

Mask for packed-decimal-integer encoding.

This subfield occupies bit positions 24 through 27 within the *Encoding* field.

MQENC_FLOAT_MASK

Mask for floating-point encoding.

This subfield occupies bit positions 20 through 23 within the *Encoding* field.

MQENC_RESERVED_MASK

Mask for reserved bits.

This subfield occupies bit positions 0 through 19 within the *Encoding* field.

Binary-integer encoding

The following values are valid for the binary-integer encoding:

MQENC_INTEGER_UNDEFINED

Binary integers are represented using an encoding that is undefined.

MQENC_INTEGER_NORMAL

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

MQENC_INTEGER_REVERSED

Binary integers are represented in the same way as MQENC_INTEGER_NORMAL, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as MQENC_INTEGER_NORMAL.

Packed-decimal-integer encoding

The following values are valid for the packed-decimal-integer encoding:

MQENC_DECIMAL_UNDEFINED

Packed-decimal integers are represented using an encoding that is undefined.

MQENC_DECIMAL_NORMAL

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 that contains the least significant decimal digit. Within that byte, the most significant four bits contain the least significant decimal digit, and the least significant four bits contain the sign. The sign is either X'C' (positive), X'D' (negative), or X'F' (unsigned).
- The least significant byte in the number has the highest address of any of the bytes in the number; the most significant byte has the lowest address.
- The least significant bit in each byte is adjacent to the byte with the next higher address; the most significant bit in each byte is adjacent to the byte with the next lower address.

MQENC_DECIMAL_REVERSED

Packed-decimal integers are represented in the same way as MQENC_DECIMAL_NORMAL, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as MQENC_DECIMAL_NORMAL.

Floating-point encoding

The following values are valid for the floating-point encoding:

MQENC_FLOAT_UNDEFINED

Floating-point numbers are represented using an encoding that is undefined.

MQENC_FLOAT_IEEE_NORMAL

Floating-point numbers are represented using the standard IEEE³ 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

3. The Institute of Electrical and Electronics Engineers

- 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 can be found in IEEE Standard 754.

MQENC_FLOAT_IEEE_REVERSED

Floating-point numbers are represented in the same way as MQENC_FLOAT_IEEE_NORMAL, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as MQENC_FLOAT_IEEE_NORMAL.

MQENC_FLOAT_S390

Floating-point numbers are represented using the standard System/390 floating-point format; this is also used by System/370™.

Constructing encodings

To construct a value for the *Encoding* field in MQMD, the relevant constants that describe the required encodings can be:

- Added together, or
- Combined using the bitwise OR operation (if the programming language supports bit operations)

Whichever method is used, combine only one of the MQENC_INTEGER_* encodings with one of the MQENC_DECIMAL_* encodings and one of the MQENC_FLOAT_* encodings.

Analyzing encodings

The *Encoding* field contains subfields; because of this, applications that need to examine the integer, packed decimal, or float encoding must use one of the techniques described below.

Using bit operations

If the programming language supports bit operations, perform the following steps:

1. Select one of the following values, according to the type of encoding required:
 - MQENC_INTEGER_MASK for the binary integer encoding
 - MQENC_DECIMAL_MASK for the packed decimal integer encoding
 - MQENC_FLOAT_MASK for the floating point encoding

Call the value A.

2. Combine the *Encoding* field with A using the bitwise AND operation; call the result B.
3. B is the encoding required, and can be tested for equality with each of the values that is valid for that type of encoding.

Using arithmetic

If the programming language *does not* support bit operations, perform the following steps using integer arithmetic:

1. Select one of the following values, according to the type of encoding required:
 - 1 for the binary integer encoding

- 16 for the packed decimal integer encoding
- 256 for the floating point encoding

Call the value A.

2. Divide the value of the *Encoding* field by A; call the result B.
3. Divide B by 16; call the result C.
4. Multiply C by 16 and subtract from B; call the result D.
5. Multiply D by A; call the result E.
6. E is the encoding required, and can be tested for equality with each of the values that is valid for that type of encoding.

Summary of machine architecture encodings

Encodings for machine architectures are shown in Table 101.

Table 101. Summary of encodings for machine architectures

Machine architecture	Binary integer encoding	Packed-decimal integer encoding	Floating-point encoding
i5/OS	normal	normal	IEEE normal
Intel® x86	reversed	reversed	IEEE reversed
PowerPC®	normal	normal	IEEE normal
System/390	normal	normal	System/390

Chapter 7. Report options and message flags

This appendix describes the *Report* and *MsgFlags* fields that are part of the message descriptor MQMD specified on the MQGET, MQPUT, and MQPUT1 calls (see “MQMD – Message descriptor” on page 181). The appendix describes:

- The structure of the report field and how the queue manager processes it
- How an application analyzes the report field
- The structure of the message-flags field

Structure of the report field

The *Report* field is a 32-bit integer that is divided into three separate subfields. These subfields identify:

- Report options that are rejected if the local queue manager does not recognize them
- Report options that are always accepted, even if the local queue manager does not recognize them
- Report options that are accepted only if certain other conditions are satisfied

Each subfield is identified by a bit mask that has 1-bits in the positions corresponding to the subfield, and 0-bits elsewhere. The bits in a subfield are not necessarily adjacent. The bits are numbered such that bit 0 is the most significant bit, and bit 31 the least significant bit. The following masks are defined to identify the subfields:

MQRO_REJECT_UNSUP_MASK

This mask identifies the bit positions within the *Report* field where report options that are not supported by the local queue manager cause the MQPUT or MQPUT1 call to fail with completion code MQCC_FAILED and reason code MQRC_REPORT_OPTIONS_ERROR.

This subfield occupies bit positions 3, and 11 through 13.

MQRO_ACCEPT_UNSUP_MASK

This mask identifies the bit positions within the *Report* field where report options that are not supported by the local queue manager are nevertheless accepted on the MQPUT or MQPUT1 calls. Completion code MQCC_WARNING with reason code MQRC_UNKNOWN_REPORT_OPTION are returned in this case.

This subfield occupies bit positions 0 through 2, 4 through 10, and 24 through 31.

The following report options are included in this subfield:

- MQRO_ACTIVITY
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_DEAD_LETTER_Q
- MQRO_DISCARD_MSG
- MQRO_EXCEPTION
- MQRO_EXCEPTION_WITH_DATA
- MQRO_EXCEPTION_WITH_FULL_DATA

- MQRO_EXPIRATION
- MQRO_EXPIRATION_WITH_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_NAN
- MQRO_NEW_MSG_ID
- MQRO_NONE
- MQRO_PAN
- MQRO_PASS_CORREL_ID
- MQRO_PASS_MSG_ID

MQRO_ACCEPT_UNSUP_IF_XMIT_MASK

This mask identifies the bit positions within the *Report* field where report options that are not supported by the local queue manager are nevertheless accepted on the MQPUT or MQPUT1 calls *provided* that both of the following conditions are satisfied:

- The message is destined for a remote queue manager.
- The application is not putting the message directly on a local transmission queue (that is, the queue identified by the *ObjectQMGrName* and *ObjectName* fields in the object descriptor specified on the MQOPEN or MQPUT1 call is not a local transmission queue).

Completion code MQCC_WARNING with reason code MQRC_UNKNOWN_REPORT_OPTION are returned if these conditions are satisfied, and MQCC_FAILED with reason code MQRC_REPORT_OPTIONS_ERROR if not.

This subfield occupies bit positions 14 through 23.

The following report options are included in this subfield:

- MQRO_COA
- MQRO_COA_WITH_DATA
- MQRO_COA_WITH_FULL_DATA
- MQRO_COD
- MQRO_COD_WITH_DATA
- MQRO_COD_WITH_FULL_DATA

If any options are specified in the *Report* field that the queue manager does not recognize, the queue manager checks each subfield in turn by using the bitwise AND operation to combine the *Report* field with the mask for that subfield. If the result of that operation is not zero, the completion code and reason codes described above are returned.

If MQCC_WARNING is returned, it is not defined which reason code is returned if other warning conditions exist.

The ability to specify and have accepted report options that are not recognized by the local queue manager is useful when sending a message with a report option that is recognized and processed by a *remote* queue manager.

Analyzing the report field

The *Report* field contains subfields; because of this, applications that need to check whether the sender of the message requested a particular report must use one of the techniques described below.

Using bit operations

If the programming language supports bit operations, perform the following steps:

1. Select one of the following values, according to the type of report to be checked:
 - MQRO_COA_WITH_FULL_DATA for COA report
 - MQRO_COD_WITH_FULL_DATA for COD report
 - MQRO_EXCEPTION_WITH_FULL_DATA for exception report
 - MQRO_EXPIRATION_WITH_FULL_DATA for expiration report

Call the value A.

On z/OS, use the MQRO*_WITH_DATA values instead of the MQRO*_WITH_FULL_DATA values.

2. Combine the *Report* field with A using the bitwise AND operation; call the result B.
3. Test B for equality with each value that is possible for that type of report. For example, if A is MQRO_EXCEPTION_WITH_FULL_DATA, test B for equality with each of the following to determine what was specified by the sender of the message:
 - MQRO_NONE
 - MQRO_EXCEPTION
 - MQRO_EXCEPTION_WITH_DATA
 - MQRO_EXCEPTION_WITH_FULL_DATA

The tests can be performed in whatever order is most convenient for the application logic.

Use a similar method to test for the MQRO_PASS_MSG_ID or MQRO_PASS_CORREL_ID options; select as the value A whichever of these two constants is appropriate, and then proceed as described above.

Using arithmetic

If the programming language *does not* support bit operations, perform the following steps using integer arithmetic:

1. Select one of the following values, according to the type of report to be checked:
 - MQRO_COA for COA report
 - MQRO_COD for COD report
 - MQRO_EXCEPTION for exception report
 - MQRO_EXPIRATION for expiration report

Call the value A.

2. Divide the *Report* field by A; call the result B.
3. Divide B by 8; call the result C.

4. Multiply C by 8 and subtract from B; call the result D.
5. Multiply D by A; call the result E.
6. Test E for equality with each value that is possible for that type of report.
For example, if A is MQRO_EXCEPTION, test E for equality with each of the following to determine what was specified by the sender of the message:
 - MQRO_NONE
 - MQRO_EXCEPTION
 - MQRO_EXCEPTION_WITH_DATA
 - MQRO_EXCEPTION_WITH_FULL_DATA

The tests can be performed in whatever order is most convenient for the application logic.

The following pseudocode illustrates this technique for exception report messages:

```
A = MQRO_EXCEPTION
B = Report/A
C = B/8
D = B - C*8
E = D*A
```

Use a similar method to test for the MQRO_PASS_MSG_ID or MQRO_PASS_CORREL_ID options; select as the value A whichever of these two constants is appropriate, and then proceed as described above, but replacing the value 8 in the steps above by the value 2.

Structure of the message-flags field

The *MsgFlags* field is a 32-bit integer that is divided into three separate subfields. These subfields identify:

- Message flags that are rejected if the local queue manager does not recognize them
- Message flags that are always accepted, even if the local queue manager does not recognize them
- Message flags that are accepted only if certain other conditions are satisfied

Note: All subfields in *MsgFlags* are reserved for use by the queue manager.

Each subfield is identified by a bit mask that has 1-bits in the positions corresponding to the subfield, and 0-bits elsewhere. The bits are numbered such that bit 0 is the most significant bit, and bit 31 the least significant bit. The following masks are defined to identify the subfields:

MQMF_REJECT_UNSUP_MASK

This mask identifies the bit positions within the *MsgFlags* field where message flags that are not supported by the local queue manager cause the MQPUT or MQPUT1 call to fail with completion code MQCC_FAILED and reason code MQRC_MSG_FLAGS_ERROR.

This subfield occupies bit positions 20 through 31.

The following message flags are included in this subfield:

- MQMF_LAST_MSG_IN_GROUP
- MQMF_LAST_SEGMENT
- MQMF_MSG_IN_GROUP

- MQMF_SEGMENT
- MQMF_SEGMENTATION_ALLOWED
- MQMF_SEGMENTATION_INHIBITED

MQMF_ACCEPT_UNSUP_MASK

This mask identifies the bit positions within the *MsgFlags* field where message flags that are not supported by the local queue manager are nevertheless accepted on the MQPUT or MQPUT1 calls. The completion code is MQCC_OK.

This subfield occupies bit positions 0 through 11.

MQMF_ACCEPT_UNSUP_IF_XMIT_MASK

This mask identifies the bit positions within the *MsgFlags* field where message flags that are not supported by the local queue manager are nevertheless accepted on the MQPUT or MQPUT1 calls *provided* that both of the following conditions are satisfied:

- The message is destined for a remote queue manager.
- The application is not putting the message directly on a local transmission queue (that is, the queue identified by the *ObjectQMGrName* and *ObjectName* fields in the object descriptor specified on the MQOPEN or MQPUT1 call is not a local transmission queue).

Completion code MQCC_OK is returned if these conditions are satisfied, and MQCC_FAILED with reason code MQRC_MSG_FLAGS_ERROR if not.

This subfield occupies bit positions 12 through 19.

If there are flags specified in the *MsgFlags* field that the queue manager does not recognize, the queue manager checks each subfield in turn by using the bitwise AND operation to combine the *MsgFlags* field with the mask for that subfield. If the result of that operation is not zero, the completion code and reason codes described above are returned.

Chapter 8. Data conversion

This collection of topics describes the interface to the data-conversion exit, and the processing performed by the queue manager when data conversion is required.

For more information about data conversion, see the document *Data Conversion under WebSphere MQ* at <http://www.ibm.com/support/docview.wss?uid=swg27005729>.

The data-conversion exit is invoked as part of the processing of the MQGET call in order to convert the application message data to the representation required by the receiving application. Conversion of the application message data is optional; it requires the MQGMO_CONVERT option to be specified on the MQGET call.

The following subjects are described:

- The processing performed by the queue manager in response to the MQGMO_CONVERT option; see “Conversion processing.”
- Processing conventions used by the queue manager when processing a built-in format; these conventions are recommended for user-written exits too. See “Processing conventions” on page 678.
- Special considerations for converting report messages; see “Conversion of report messages” on page 683.
- The parameters passed to the data-conversion exit; see “MQ_DATA_CONV_EXIT – Data conversion exit” on page 697.
- A call that can be used from the exit to convert character data between different representations; see “MQXCNVC – Convert characters” on page 691.
- The data-structure parameter that is specific to the exit; see “MQDXP – Data-conversion exit parameter” on page 684.

Conversion processing

The queue manager performs the following actions if the MQGMO_CONVERT option is specified on the MQGET call, and there is a message to be returned to the application:

1. If one or more of the following is true, no conversion is necessary:
 - The message data is already in the character set and encoding required by the application issuing the MQGET call. The application must set the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter of the MQGET call to the values required, before issuing the call.
 - The length of the message data is zero.
 - The length of the *Buffer* parameter of the MQGET call is zero.

In these cases the message is returned without conversion to the application issuing the MQGET call; the *CodedCharSetId* and *Encoding* values in the *MsgDesc* parameter are set to the values in the control information in the message, and the call completes with one of the following combinations of completion code and reason code:

Completion code	Reason code
MQCC_OK	MQRC_NONE

Completion code	Reason code
MQCC_WARNING	MQRC_TRUNCATED_MSG_ACCEPTED
MQCC_WARNING	MQRC_TRUNCATED_MSG_FAILED

The following steps are performed only if the character set or encoding of the message data differs from the corresponding value in the *MsgDesc* parameter, and there is data to be converted:

- If the *Format* field in the control information in the message has the value MQFMT_NONE, the message is returned unconverted, with completion code MQCC_WARNING and reason code MQRC_FORMAT_ERROR.
In all other cases conversion processing continues.
 - The message is removed from the queue and placed in a temporary buffer that 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.
 - If the message has to be truncated to fit in the buffer, the following is done:
 - If the MQGMO_ACCEPT_TRUNCATED_MSG option was *not* specified, the message is returned unconverted, with completion code MQCC_WARNING and reason code MQRC_TRUNCATED_MSG_FAILED.
 - If the MQGMO_ACCEPT_TRUNCATED_MSG option *was* specified, the completion code is set to MQCC_WARNING, the reason code is set to MQRC_TRUNCATED_MSG_ACCEPTED, and conversion processing continues.
 - If the message can be accommodated in the buffer without truncation, or the MQGMO_ACCEPT_TRUNCATED_MSG option was specified, the following is done:
 - If the format is a built-in format, the buffer is passed to the queue-manager's data-conversion service.
 - If the format is not a built-in format, the buffer is passed to a user-written exit with the same name as the format. If the exit cannot be found, the message is returned unconverted, with completion code MQCC_WARNING and reason code MQRC_FORMAT_ERROR.
- If no error occurs, the output from the data-conversion service or from the user-written exit is the converted message, plus the completion code and reason code to be returned to the application issuing the MQGET call.
- If the conversion is successful, the queue manager returns the converted message to the application. In this case, the completion code and reason code returned by the MQGET call are one of the following combinations:

Completion code	Reason code
MQCC_OK	MQRC_NONE
MQCC_WARNING	MQRC_TRUNCATED_MSG_ACCEPTED

However, if the conversion is performed by a user-written exit, other reason codes can be returned, even when the conversion is successful.

If the conversion fails, the queue manager returns the unconverted message to the application, with the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter set to the values in the control information in the message, and with completion code MQCC_WARNING. See below for possible reason codes.

Processing conventions

When converting a built-in format, the queue manager follows the processing conventions described below.

User-written exits should also follow these conventions, although this is not enforced by the queue manager. The built-in formats converted by the queue manager are:

- MQFMT_ADMIN
- MQFMT_CICS (z/OS only)
- MQFMT_COMMAND_1
- MQFMT_COMMAND_2
- MQFMT_DEAD_LETTER_HEADER
- MQFMT_DIST_HEADER
- MQFMT_EVENT version 1
- MQFMT_EVENT version 2 (z/OS only)
- MQFMT_IMS
- MQFMT_IMS_VAR_STRING
- MQFMT_MD_EXTENSION
- MQFMT_PCF
- MQFMT_REF_MSG_HEADER
- MQFMT_RF_HEADER
- MQFMT_RF_HEADER_2
- MQFMT_STRING
- MQFMT_TRIGGER
- MQFMT_WORK_INFO_HEADER (z/OS only)
- MQFMT_XMIT_Q_HEADER

1. If the message expands during conversion, and exceeds the size of the *Buffer* parameter, the following is done:
 - If the MQGMO_ACCEPT_TRUNCATED_MSG option was *not* specified, the message is returned unconverted, with completion code MQCC_WARNING and reason code MQRC_CONVERTED_MSG_TOO_BIG.
 - If the MQGMO_ACCEPT_TRUNCATED_MSG option *was* specified, the message is truncated, the completion code is set to MQCC_WARNING, the reason code is set to MQRC_TRUNCATED_MSG_ACCEPTED, and conversion processing continues.
2. If truncation occurs (either before or during conversion), the number of valid bytes returned in the *Buffer* parameter can 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 those bytes in the returned message do not contain valid information. This can also occur if a message that was truncated before conversion shrinks during conversion.

If the number of valid bytes returned is less than the length of the buffer, the unused bytes at the end of the buffer are set to nulls.

3. If an array or string straddles the end of the buffer, as much of the data as possible is converted; only the particular array element or DBCS character which is incomplete is not converted; preceding array elements or characters are converted.
4. If truncation occurs (either before or during conversion), the length returned for the *DataLength* parameter is the length of the *unconverted* message before truncation.

5. When strings are converted between single-byte character sets (SBCS), double-byte character sets (DBCS), or multi-byte character sets (MBCS), the strings can expand or contract.
 - In the PCF formats MQFMT_ADMIN, MQFMT_EVENT, and MQFMT_PCF, the strings in the MQCFST and MQCFSL structures expand or contract as necessary to accommodate the string after conversion.

For the string-list structure MQCFSL, the strings in the list might expand or contract by different amounts. If this happens, the queue manager pads the shorter strings with blanks to make them the same length as the longest string after conversion.
 - In the format MQFMT_REF_MSG_HEADER, the strings addressed by the *SrcEnvOffset*, *SrcNameOffset*, *DestEnvOffset*, and *DestNameOffset* fields expand or contract as necessary to accommodate the strings after conversion.
 - In the format MQFMT_RF_HEADER, the *NameValueString* field expands or contracts as necessary to accommodate the name/value pairs after conversion.
 - In structures with fixed field sizes, the queue manager allows strings to expand or contract within their fixed fields, provided that no significant information is lost. In this regard, trailing blanks and characters following the first null character in the field are treated as insignificant.
 - If the string expands, but only insignificant characters need to be discarded to accommodate the converted string in the field, the conversion succeeds and the call completes with MQCC_OK and reason code MQRC_NONE (assuming no other errors).
 - If the string expands, but the converted string requires significant characters to be discarded in order to fit in the field, the message is returned unconverted and the call completes with MQCC_WARNING and reason code MQRC_CONVERTED_STRING_TOO_BIG.

Note: Reason code MQRC_CONVERTED_STRING_TOO_BIG results in this case whether or not the MQGMO_ACCEPT_TRUNCATED_MSG option was specified.

 - If the string contracts, the queue manager pads the string with blanks to the length of the field.
6. For messages consisting of one or more MQ header structures followed by user data, one or more of the header structures might be converted, while the remainder of the message is not. However, (with two exceptions) the *CodedCharSetId* and *Encoding* fields in each header structure always correctly indicate the character set and encoding of the data that follows the header structure.

The two exceptions are the MQCIH and MQIIH structures, where the values in the *CodedCharSetId* and *Encoding* fields in those structures are not significant. For those structures, the data following the structure is in the same character set and encoding as the MQCIH or MQIIH structure itself.
7. If the *CodedCharSetId* or *Encoding* fields in the control information of the message being retrieved, or in the *MsgDesc* parameter, specify values that are undefined or not supported, the queue manager might ignore the error if the undefined or unsupported value does not need to be used in converting the message.

For example, if the *Encoding* field in the message specifies an unsupported float encoding, but the message contains only integer data, or contains

floating-point data that does not require conversion (because the source and target float encodings are identical), the error might not be diagnosed.

If the error is diagnosed, the message is returned unconverted, with completion code MQCC_WARNING and one of the MQRC_SOURCE_*_ERROR or MQRC_TARGET_*_ERROR reason codes (as appropriate); the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to the values in the control information in the message.

If the error is not diagnosed and the conversion completes successfully, the values returned in the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are those specified by the application issuing the MQGET call.

8. In all cases, if the message is returned to the application unconverted the completion code is set to MQCC_WARNING, and the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to the values appropriate to the unconverted data. This is done for MQFMT_NONE also.

The *Reason* parameter is set to a code that indicates why the conversion could not be carried out, unless the message also had to be truncated; reason codes related to truncation take precedence over reason codes related to conversion. (To determine if a truncated message was converted, check the values returned in the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter.)

When an error is diagnosed, either a specific reason code is returned, or the general reason code MQRC_NOT_CONVERTED. The reason code returned depends on the diagnostic capabilities of the underlying data-conversion service.

9. If completion code MQCC_WARNING is returned, and more than one reason code is relevant, the order of precedence is as follows:
 - a. The following reasons take precedence over all others; only one of the reasons in this group can arise:
 - MQRC_SIGNAL_REQUEST_ACCEPTED
 - MQRC_TRUNCATED_MSG_ACCEPTED
 - b. The order of precedence within the remaining reason codes is not defined.
10. On completion of the MQGET call:
 - The following reason code indicates that the message was converted successfully:
 - MQRC_NONE
 - The following reason codes indicate that the message *might* have been converted successfully (check the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter to find out):
 - MQRC_MSG_MARKED_BROWSE_CO_OP
 - MQRC_TRUNCATED_MSG_ACCEPTED
 - All other reason codes indicate that the message was not converted.

The following processing is specific to the built-in formats; it does not apply to user-defined formats:

11. With the exception of the following formats:
 - MQFMT_ADMIN
 - MQFMT_COMMAND_1
 - MQFMT_COMMAND_2
 - MQFMT_EVENT
 - MQFMT_IMS_VAR_STRING
 - MQFMT_PCF

- MQFMT_STRING

none of the built-in formats can be converted from or to character sets that do not have SBCS characters for the characters that are valid in queue names. If an attempt is made to perform such a conversion, the message is returned unconverted, with completion code MQCC_WARNING and reason code MQRC_SOURCE_CCSD_ERROR or MQRC_TARGET_CCSD_ERROR, as appropriate.

The Unicode character set UCS-2 is an example of a character set that does not have SBCS characters for the characters that are valid in queue names.

12. If the message data for a built-in format is truncated, fields within the message that contain lengths of strings, or counts of elements or structures, are *not* adjusted to reflect the length of the data actually returned to the application; the values returned for such fields within the message data are the values applicable to the message *prior to truncation*.

When processing messages such as a truncated MQFMT_ADMIN message, ensure that the application does not attempt to access data beyond the end of the data returned.

13. If the format name is MQFMT_DEAD_LETTER_HEADER, the message data begins with an MQDLH structure, possibly followed by zero or more bytes of application message data. The format, character set, and encoding of the application message data are defined by the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQDLH structure at the start of the message. Because the MQDLH structure and application message data can have different character sets and encodings, one, other, or both of the MQDLH structure and application message data might require conversion.

The queue manager converts the MQDLH structure first, as necessary. If conversion is successful, or the MQDLH structure does not require conversion, the queue manager checks the *CodedCharSetId* and *Encoding* fields in the MQDLH structure to see if conversion of the application message data is required. If conversion *is* required, the queue manager invokes the user-written exit with the name given by the *Format* field in the MQDLH structure, or performs the conversion itself (if *Format* is the name of a built-in format).

If the MQGET call returns a completion code of MQCC_WARNING, and the reason code is one of those indicating that conversion was not successful, one of the following applies:

- The MQDLH structure could not be converted. In this case the application message data will not have been converted either.
- The MQDLH structure was converted, but the application message data was not.

The application can examine the values returned in the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter, and those in the MQDLH structure, in order to determine which of the above applies.

14. If the format name is MQFMT_XMIT_Q_HEADER, the message data begins with an MQXQH structure, possibly followed by zero or more bytes of additional data. This additional data is usually the application message data (which may be of zero length), but there can also be one or more further MQ header structures present, at the start of the additional data.

The MQXQH structure must be in the character set and encoding of the queue manager. The format, character set, and encoding of the data following the MQXQH structure are given by the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQMD structure contained *within* the MQXQH. For each

subsequent MQ header structure present, the *Format*, *CodedCharSetId*, and *Encoding* fields in the structure describe the data that follows that structure; that data is either another MQ header structure, or the application message data.

If the MQGMO_CONVERT option is specified for an MQFMT_XMIT_Q_HEADER message, the application message data and certain of the MQ header structures are converted, *but the data in the MQXQH structure is not*. On return from the MQGET call, therefore:

- The values of the *Format*, *CodedCharSetId*, and *Encoding* fields in the *MsgDesc* parameter describe the data in the MQXQH structure, and *not* the application message data; the values are therefore *not* the same as those specified by the application that issued the MQGET call.

The effect of this is that an application that repeatedly gets messages from a transmission queue with the MQGMO_CONVERT option specified must reset the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter to the values required for the application message data, prior to each MQGET call.

- The values of the *Format*, *CodedCharSetId*, and *Encoding* fields in the last MQ header structure present describe the application message data. If there are no other MQ header structures present, the application message data is described by these fields in the MQMD structure within the MQXQH structure. If conversion is successful, the values will be the same as those specified in the *MsgDesc* parameter by the application that issued the MQGET call.

If the message is a distribution-list message, the MQXQH structure is followed by an MQDH structure (plus its arrays of MQOR and MQPMR records), which in turn might be followed by zero or more further MQ header structures and zero or more bytes of application message data. Like the MQXQH structure, the MQDH structure must be in the character set and encoding of the queue manager, and it is not converted on the MQGET call, even if the MQGMO_CONVERT option is specified.

The processing of the MQXQH and MQDH structures described above is primarily intended for use by message channel agents when they get messages from transmission queues.

Conversion of report messages

In general a report message can contain varying amounts of application message data, according to the report options specified by the sender of the original message. However, an activity report can contain data but without the report option mentioning *_WITH_DATA in the constant.

In particular, a report message can contain either:

1. No application message data
2. Some of the application message data from the original message
This occurs when the sender of the original message specifies MQRO_*_WITH_DATA and the message is longer than 100 bytes.
3. All the application message data from the original message
This occurs when the sender of the original message specifies MQRO_*_WITH_FULL_DATA, or specifies MQRO_*_WITH_DATA and the message is 100 bytes or shorter.

When the queue manager or message channel agent generates a report message, it copies the format name from the original message into the *Format* field in the

control information in the report message. The format name in the report message might therefore imply a length of data that is different from the length actually present in the report message (cases 1 and 2 above).

If the MQGMO_CONVERT option is specified when the report message is retrieved:

- For case 1 above, the data-conversion exit is not invoked (because the report message has 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 is invoked to convert a message that is *shorter* than the length implied by the format name.

In addition, the reason code passed to the exit is usually MQRC_NONE (that is, the reason code does 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 must *not* use the format name to deduce the length of data passed to it; instead the exit must check the length of data provided, and be prepared to convert *less* data than the length implied by the format name. If the data can be converted successfully, completion code MQCC_OK and reason code MQRC_NONE must be returned by the exit. The length of the message data to be converted is passed to the exit as the *InBufferLength* parameter.

Product-sensitive programming interface

MQDXP – Data-conversion exit parameter

The following table summarizes the fields in the structure.

Table 102. Fields in MQDXP

Field	Description	Topic
<i>StrucId</i>	Structure identifier	StrucId
<i>Version</i>	Structure version number	Version
<i>AppOptions</i>	Application options	AppOptions
<i>Encoding</i>	Numeric encoding required by application	Encoding
<i>CodedCharSetId</i>	Character set required by application	CodedCharSetId
<i>DataLength</i>	Length in bytes of message data	DataLength
<i>CompCode</i>	Completion code	CompCode
<i>Reason</i>	Reason code qualifying <i>CompCode</i>	Reason
<i>ExitResponse</i>	Response from exit	ExitResponse
<i>Hconn</i>	Connection handle	Hconn

Overview

Purpose: The MQDXP structure is a parameter that the queue manager passes to the data-conversion exit when the exit is invoked to convert the message data as

part of the processing of the MQGET call. See the description of the MQ_DATA_CONV_EXIT call for details of the data conversion exit.

Character set and encoding: Character data in MQDXP is in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute. Numeric data in MQDXP is in the native machine encoding; this is given by MQENC_NATIVE.

Usage: Only the *DataLength*, *CompCode*, *Reason*, and *ExitResponse* fields in MQDXP can be changed by the exit; changes to other fields are ignored. However, the *DataLength* field *cannot* be changed if the message being converted is a segment that contains only part of a logical message.

When control returns to the queue manager from the exit, the queue manager checks the values returned in MQDXP. If the values returned are not valid, the queue manager continues processing as though the exit had returned MQXDR_CONVERSION_FAILED in *ExitResponse*; however, the queue manager ignores the values of the *CompCode* and *Reason* fields returned by the exit in this case, and uses instead the values those fields had on *input* to the exit. The following values in MQDXP cause this processing to occur:

- *ExitResponse* field not MQXDR_OK and not MQXDR_CONVERSION_FAILED
- *CompCode* field not MQCC_OK and not MQCC_WARNING
- *DataLength* field less than zero, or *DataLength* field changed when the message being converted is a segment that contains only part of a logical message.

Fields

The MQDXP structure contains the following fields; the fields are described in alphabetic order:

AppOptions (MQLONG)

This is a copy of the *Options* field of the MQGMO structure specified by the application issuing the MQGET call. The exit might need to examine these to ascertain whether the MQGMO_ACCEPT_TRUNCATED_MSG option was specified.

This is an input field to the exit.

CodedCharSetId (MQLONG)

This is the coded character-set identifier of the character set required by the application issuing the MQGET call; see the *CodedCharSetId* field in the MQMD structure for more details. If the application specifies the special value MQCCSI_Q_MGR on the MQGET call, the queue manager changes this to the actual character-set identifier of the character set used by the queue manager, before invoking the exit.

If the conversion is successful, the exit must copy this to the *CodedCharSetId* field in the message descriptor.

This is an input field to the exit.

CompCode (MQLONG)

When the exit is invoked, this contains the completion code that is returned to the application that issued the MQGET call, if the exit chooses to do nothing. It is always MQCC_WARNING, because either the message was truncated, or the message requires conversion and this has not yet been done.

On output from the exit, this field contains the completion code to be returned to the application in the *CompCode* parameter of the MQGET call; only MQCC_OK and MQCC_WARNING are valid. See the description of the *Reason* field for recommendations on how the exit should set this field on output.

This is an input/output field to the exit.

DataLength (MQLONG)

When the exit is invoked, this field contains the original length of the application message data. If the message was truncated to fit into the buffer provided by the application, the size of the message provided to the exit will be *smaller* than the value of *DataLength*. The size of the message actually provided to the exit is always given by the *InBufferLength* parameter of the exit, irrespective of any truncation that may have occurred.

Truncation is indicated by the *Reason* field having the value MQRC_TRUNCATED_MSG_ACCEPTED on input to the exit.

Most conversions will not need to change this length, but an exit can do so if necessary; the value set by the exit is returned to the application in the *DataLength* parameter of the MQGET call. However, this length *cannot* be changed if the message being converted is a segment that contains only part of a logical message. This is because changing the length would cause the offsets of later segments in the logical message to be incorrect.

Note that, if the exit wants to change the length of the data, be aware that the queue manager has already decided whether the message data will fit into the application's buffer, based on the length of the *unconverted* data. This decision determines whether the message is removed from the queue (or the browse cursor moved, for a browse request), and is not affected by any change to the data length caused by the conversion. For this reason it is recommended that conversion exits do not cause a change in the length of the application message data.

If character conversion does imply a change of length, a string can be converted into another string with the same length in bytes, truncating trailing blanks or padding with blanks as necessary.

The exit is not invoked if the message contains no application message data; hence *DataLength* is always greater than zero.

This is an input/output field to the exit.

Encoding (MQLONG)

Numeric encoding required by application.

This is the numeric encoding required by the application issuing the MQGET call; see the *Encoding* field in the MQMD structure for more details.

If the conversion is successful, the exit should copy this to the *Encoding* field in the message descriptor.

This is an input field to the exit.

ExitOptions (MQLONG)

Reserved.

This is a reserved field; its value is 0.

ExitResponse (MQLONG)

Response from exit.

This is set by the exit to indicate the success or otherwise of the conversion. It must be one of the following:

MQXDR_OK

Conversion was successful.

If the exit specifies this value, the queue manager returns the following to the application that issued the MQGET call:

- The value of the *CompCode* field on output from the exit
- The value of the *Reason* field on output from the exit
- The value of the *DataLength* field on output from the exit
- The contents of the exit's output buffer *OutBuffer*. The number of bytes returned is the lesser of the exit's *OutBufferLength* parameter, and the value of the *DataLength* field on output from the exit.

If the *Encoding* and *CodedCharSetId* fields in the exit's message descriptor parameter are *both* unchanged, the queue manager returns:

- The value of the *Encoding* and *CodedCharSetId* fields in the MQDXP structure on *input* to the exit.

If one or both of the *Encoding* and *CodedCharSetId* fields in the exit's message descriptor parameter has been changed, the queue manager returns:

- The value of the *Encoding* and *CodedCharSetId* fields in the exit's message descriptor parameter on output from the exit

MQXDR_CONVERSION_FAILED

Conversion was unsuccessful.

If the exit specifies this value, the queue manager returns the following to the application that issued the MQGET call:

- The value of the *CompCode* field on output from the exit
- The value of the *Reason* field on output from the exit
- The value of the *DataLength* field on *input* to the exit
- The contents of the exit's input buffer *InBuffer*. The number of bytes returned is given by the *InBufferLength* parameter

If the exit has altered *InBuffer*, the results are undefined.

ExitResponse is an output field from the exit.

Hconn (MQHCONN)

Connection handle.

This is a connection handle which can be used on the MQXCNVC call. This handle is not necessarily the same as the handle specified by the application which issued the MQGET call.

Reason (MQLONG)

Reason code qualifying *CompCode*.

When the exit is invoked, this contains the reason code that will be returned to the application that issued the MQGET call, if the exit chooses to do nothing. Among possible values are MQRC_TRUNCATED_MSG_ACCEPTED, indicating that the message was truncated in order fit into the buffer provided by the application, and MQRC_NOT_CONVERTED, indicating that the message requires conversion but that this has not yet been done.

On output from the exit, this field contains the reason to be returned to the application in the *Reason* parameter of the MQGET call; the following is recommended:

- If *Reason* had the value MQRC_TRUNCATED_MSG_ACCEPTED on input to the exit, the *Reason* and *CompCode* fields should not be altered, irrespective of whether the conversion succeeds or fails.

(If the *CompCode* field is not MQCC_OK, the application which retrieves the message can identify a conversion failure by comparing the returned *Encoding* and *CodedCharSetId* values in the message descriptor with the values requested; in contrast, the application cannot distinguish a truncated message from a message that just fitted the buffer. For this reason, MQRC_TRUNCATED_MSG_ACCEPTED should be returned in preference to any of the reasons that indicate conversion failure.)

- If *Reason* had any other value on input to the exit:
 - If the conversion succeeds, *CompCode* should be set to MQCC_OK and *Reason* set to MQRC_NONE.
 - If the conversion fails, or the message expands and has to be truncated to fit in the buffer, *CompCode* should be set to MQCC_WARNING (or left unchanged), and *Reason* set to one of the values listed below, to indicate the nature of the failure.

Note that, if the message after conversion is too big for the buffer, it should be truncated only if the application that issued the MQGET call specified the MQGMO_ACCEPT_TRUNCATED_MSG option:

- If it did specify that option, reason MQRC_TRUNCATED_MSG_ACCEPTED should be returned.
- If it did not specify that option, the message should be returned unconverted, with reason code MQRC_CONVERTED_MSG_TOO_BIG.

The reason codes listed below are recommended for use by the exit to indicate the reason that conversion failed, but the exit can return other values from the set of MQRC_* codes if deemed appropriate. In addition, the range of values MQRC_APPL_FIRST through MQRC_APPL_LAST are allocated for use by the exit to indicate conditions that the exit wishes to communicate to the application issuing the MQGET call.

Note: If the message cannot be converted successfully, the exit *must* return MQXDR_CONVERSION_FAILED in the *ExitResponse* field, in order to cause the queue manager to return the unconverted message. This is true regardless of the reason code returned in the *Reason* field.

MQRC_APPL_FIRST
(900, X'384') Lowest value for application-defined reason code.

MQRC_APPL_LAST
(999, X'3E7') Highest value for application-defined reason code.

MQRC_CONVERTED_MSG_TOO_BIG
(2120, X'848') Converted data too big for buffer.

MQRC_NOT_CONVERTED
(2119, X'847') Message data not converted.

MQRC_SOURCE_CCSDID_ERROR
(2111, X'83F') Source coded character set identifier not valid.

MQRC_SOURCE_DECIMAL_ENC_ERROR
(2113, X'841') Packed-decimal encoding in message not recognized.

MQRC_SOURCE_FLOAT_ENC_ERROR
(2114, X'842') Floating-point encoding in message not recognized.

MQRC_SOURCE_INTEGER_ENC_ERROR
(2112, X'840') Source integer encoding not recognized.

MQRC_TARGET_CCSDID_ERROR
(2115, X'843') Target coded character set identifier not valid.

MQRC_TARGET_DECIMAL_ENC_ERROR
(2117, X'845') Packed-decimal encoding specified by receiver not recognized.

MQRC_TARGET_FLOAT_ENC_ERROR
(2118, X'846') Floating-point encoding specified by receiver not recognized.

MQRC_TARGET_INTEGER_ENC_ERROR
(2116, X'844') Target integer encoding not recognized.

MQRC_TRUNCATED_MSG_ACCEPTED
(2079, X'81F') Truncated message returned (processing completed).

This is an input/output field to the exit.

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQDXP_STRUC_ID

Identifier for data conversion exit parameter structure.

For the C programming language, the constant `MQDXP_STRUC_ID_ARRAY` is also defined; this has the same value as `MQDXP_STRUC_ID`, but is an array of characters instead of a string.

This is an input field to the exit.

Version (MQLONG)

Structure version number.

The value must be:

MQDXP_VERSION_1

Version number for data-conversion exit parameter structure.

The following constant specifies the version number of the current version:

MQDXP_CURRENT_VERSION

Current version of data-conversion exit parameter structure.

Note: When a new version of this structure is introduced, the layout of the existing part is not changed. The exit should therefore check that the *Version* field is equal to or greater than the lowest version which contains the fields that the exit needs to use.

This is an input field to the exit.

C declaration

```
typedef struct tagMQDXP MQDXP;
struct tagMQDXP {
    MQCHAR4  StrucId;           /* Structure identifier */
    MQLONG   Version;          /* Structure version number */
    MQLONG   ExitOptions;      /* Reserved */
    MQLONG   AppOptions;       /* Application options */
    MQLONG   Encoding;         /* Numeric encoding required by
                               application */
    MQLONG   CodedCharSetId;   /* Character set required by application */
    MQLONG   DataLength;       /* Length in bytes of message data */
    MQLONG   CompCode;         /* Completion code */
    MQLONG   Reason;           /* Reason code qualifying CompCode */
    MQLONG   ExitResponse;     /* Response from exit */
    MQHCONN  Hconn;           /* Connection handle */
};
```

COBOL declaration (i5/OS only)

```
** MQDXP structure
10 MQDXP.
** Structure identifier
15 MQDXP-STRUCID PIC X(4).
** Structure version number
15 MQDXP-VERSION PIC S9(9) BINARY.
** Reserved
15 MQDXP-EXITOPTIONS PIC S9(9) BINARY.
** Application options
15 MQDXP-APPOPTIONS PIC S9(9) BINARY.
** Numeric encoding required by application
15 MQDXP-ENCODING PIC S9(9) BINARY.
** Character set required by application
15 MQDXP-CODEDCHARSETID PIC S9(9) BINARY.
** Length in bytes of message data
15 MQDXP-DATALength PIC S9(9) BINARY.
** Completion code
15 MQDXP-COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
15 MQDXP-REASON PIC S9(9) BINARY.
** Response from exit
15 MQDXP-EXITRESPONSE PIC S9(9) BINARY.
** Connection handle
15 MQDXP-HCONN PIC S9(9) BINARY.
```

System/390 assembler declaration

```
MQDXP          DSECT
MQDXP_STRUCID  DS CL4 Structure identifier
MQDXP_VERSION  DS F   Structure version number
MQDXP_EXITOPTIONS DS F   Reserved
MQDXP_APPOPTIONS DS F   Application options
MQDXP_ENCODING DS F   Numeric encoding required by application
```

MQDXP_CODEDCHARSETID	DS	F	Character set required by application
MQDXP_DATALENGTH	DS	F	Length in bytes of message data
MQDXP_COMPCODE	DS	F	Completion code
MQDXP_REASON	DS	F	Reason code qualifying COMPCODE
MQDXP_EXITRESPONSE	DS	F	Response from exit
MQDXP_HCONN	DS	F	Connection handle
*			
MQDXP_LENGTH	EQU	*-MQDXP	
	ORG	MQDXP	
MQDXP_AREA	DS	CL(MQDXP_LENGTH)	

MQXCNVC – Convert characters

The MQXCNVC call converts characters from one character set to another using the C programming language.

Note:

1. In all environments the call can be used from a batch application as well as from a data-conversion exit.
2. The MQXCNVC call is not available from a client environment.

This call is part of the WebSphere MQ Data Conversion Interface (DCI), which is one of the WebSphere MQframework interfaces.

Syntax

MQXCNVC (Hconn, Options, SourceCCSID, SourceLength, SourceBuffer, TargetCCSID, TargetLength, TargetBuffer, DataLength, CompCode, Reason)

Parameters

The MQXCNVC call has the following parameters.

Hconn (MQHCONN) – input

Connection handle.

This handle represents the connection to the queue manager.

In a data-conversion exit, *Hconn* should normally be the handle passed to the data-conversion exit in the *Hconn* field of the MQDXP structure; this handle is not necessarily the same as the handle specified by the application which issued the MQGET call.

On i5/OS, the following special value can be specified for *Hconn*:

MQHC_DEF_HCONN

Default connection handle.

Options (MQLONG) – input

Options that control the action of MQXCNVC.

Zero or more of the options described below can be specified. If more than one is required, the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations)

Default-conversion option: The following option controls the use of default character conversion:

MQDCC_DEFAULT_CONVERSION

Default conversion.

This option specifies that default character conversion can be used if one or both of the character sets specified on the call is not supported. This allows the queue manager to use an installation-specified default character set that approximates the specified character set, when converting the string.

Note: The result of using an approximate character set to convert the string is that some characters may be converted incorrectly. This can be avoided by using in the string only characters which are common to both the specified character set and the default character set.

The default character sets are defined by a configuration option when the queue manager is installed or restarted.

If MQDCC_DEFAULT_CONVERSION is not specified, the queue manager uses only the specified character sets to convert the string, and the call fails if one or both of the character sets is not supported.

This option is supported in the following environments: AIX, HP-UX, OS/2, i5/OS, Solaris, Linux, Windows.

Padding option: The following option allows the queue manager to pad the converted string with blanks or discard insignificant trailing characters, in order to make the converted string fit the target buffer:

MQDCC_FILL_TARGET_BUFFER

Fill target buffer.

This option requests that conversion take place in such a way that the target buffer is filled completely:

- If the string contracts when it is converted, trailing blanks are added in order to fill the target buffer.
- If the string expands when it is converted, trailing characters that are not significant are discarded to make the converted string fit the target buffer. If this can be done successfully, the call completes with MQCC_OK and reason code MQRC_NONE.

If there are too few insignificant trailing characters, as much of the string as will fit is placed in the target buffer, and the call completes with MQCC_WARNING and reason code MQRC_CONVERTED_MSG_TOO_BIG.

Insignificant characters are:

- Trailing blanks
- Characters following the first null character in the string (but excluding the first null character itself)
- If the string, *TargetCCSID*, and *TargetLength* are such that the target buffer cannot be set completely with valid characters, the call fails with MQCC_FAILED and reason code MQRC_TARGET_LENGTH_ERROR. This can occur when *TargetCCSID* is a pure DBCS character set (such as UCS-2), but *TargetLength* specifies a length that is an odd number of bytes.
- *TargetLength* can be less than or greater than *SourceLength*. On return from MQXCNVC, *DataLength* has the same value as *TargetLength*.

If this option is not specified:

- The string is allowed to contract or expand within the target buffer as required. Insignificant trailing characters are neither added nor discarded.

If the converted string fits in the target buffer, the call completes with MQCC_OK and reason code MQRC_NONE.

If the converted string is too big for the target buffer, as much of the string as will fit is placed in the target buffer, and the call completes with MQCC_WARNING and reason code MQRC_CONVERTED_MSG_TOO_BIG. Note that fewer than *TargetLength* bytes can be returned in this case.

- *TargetLength* can be less than or greater than *SourceLength*. On return from MQXCNVC, *DataLength* is less than or equal to *TargetLength*.

This option is supported in the following environments: AIX, HP-UX, OS/2, i5/OS, Solaris, Linux, Windows.

Encoding options: The options described below can be used to specify the integer encodings of the source and target strings. The relevant encoding is used *only* when the corresponding character set identifier indicates that the representation of the character set in main storage is dependent on the encoding used for binary integers. This affects only certain multibyte character sets (for example, UCS-2 character sets).

The encoding is ignored if the character set is a single-byte character set (SBCS), or a multibyte character set whose representation in main storage is not dependent on the integer encoding.

Only one of the MQDCC_SOURCE_* values should be specified, combined with one of the MQDCC_TARGET_* values:

MQDCC_SOURCE_ENC_NATIVE

Source encoding is the default for the environment and programming language.

MQDCC_SOURCE_ENC_NORMAL

Source encoding is normal.

MQDCC_SOURCE_ENC_REVERSED

Source encoding is reversed.

MQDCC_SOURCE_ENC_UNDEFINED

Source encoding is undefined.

MQDCC_TARGET_ENC_NATIVE

Target encoding is the default for the environment and programming language.

MQDCC_TARGET_ENC_NORMAL

Target encoding is normal.

MQDCC_TARGET_ENC_REVERSED

Target encoding is reversed.

MQDCC_TARGET_ENC_UNDEFINED

Target encoding is undefined.

The encoding values defined above can be added directly to the *Options* field. However, if the source or target encoding is obtained from the *Encoding* field in the MQMD or other structure, the following processing must be done:

1. The integer encoding must be extracted from the *Encoding* field by eliminating the float and packed-decimal encodings; see “Analyzing encodings” on page 669 for details of how to do this.
2. The integer encoding resulting from step 1 must be multiplied by the appropriate factor before being added to the *Options* field. These factors are:
 - MQDCC_SOURCE_ENC_FACTOR for the source encoding
 - MQDCC_TARGET_ENC_FACTOR for the target encoding

The following illustrates how this might be coded in the C programming language:

```
Options = (MsgDesc.Encoding & MQENC_INTEGER_MASK)
          * MQDCC_SOURCE_ENC_FACTOR
          + (DataConvExitParms.Encoding & MQENC_INTEGER_MASK)
          * MQDCC_TARGET_ENC_FACTOR;
```

If not specified, the encoding options default to undefined (MQDCC_*_ENC_UNDEFINED). In most cases, this does not affect the successful completion of the MQXCNVC call. However, if the corresponding character set is a multibyte character set whose representation is dependent on the encoding (for example, a UCS-2 character set), the call fails with reason code MQRC_SOURCE_INTEGER_ENC_ERROR or MQRC_TARGET_INTEGER_ENC_ERROR as appropriate.

The encoding options are supported in the following environments: AIX, HP-UX, z/OS, OS/2, i5/OS, Solaris, Linux, Windows.

Default option: If none of the options described above is specified, the following option can be used:

MQDCC_NONE

No options specified.

MQDCC_NONE is defined to aid program documentation. It is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

SourceCCSID (MQLONG) – input

Coded character set identifier of string before conversion.

This is the coded character set identifier of the input string in *SourceBuffer*.

SourceLength (MQLONG) – input

Length of string before conversion.

This is the length in bytes of the input string in *SourceBuffer*; it must be zero or greater.

SourceBuffer (MQCHAR×SourceLength) – input

String to be converted.

This is the buffer containing the string to be converted from one character set to another.

TargetCCSID (MQLONG) – input

Coded character set identifier of string after conversion.

This is the coded character set identifier of the character set to which *SourceBuffer* is to be converted.

TargetLength (MQLONG) – input

Length of output buffer.

This is the length in bytes of the output buffer *TargetBuffer*; it must be zero or greater. It can be less than or greater than *SourceLength*.

TargetBuffer (MQCHAR×TargetLength) – output

String after conversion.

This is the string after it has been converted to the character set defined by *TargetCCSID*. The converted string can be shorter or longer than the unconverted string. The *DataLength* parameter indicates the number of valid bytes returned.

DataLength (MQLONG) – output

Length of output string.

This is the length of the string returned in the output buffer *TargetBuffer*. The converted string can be shorter or longer than the unconverted string.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK

Successful completion.

MQCC_WARNING

Warning (partial completion).

MQCC_FAILED

Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE

(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_CONVERTED_MSG_TOO_BIG

(2120, X'848') Converted data too big for buffer.

If *CompCode* is MQCC_FAILED:

MQRC_DATA_LENGTH_ERROR

(2010, X'7DA') Data length parameter not valid.

MQRC_DBCS_ERROR

(2150, X'866') DBCS string not valid.

MQRC_HCONN_ERROR

(2018, X'7E2') Connection handle not valid.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SOURCE_BUFFER_ERROR
(2145, X'861') Source buffer parameter not valid.

MQRC_SOURCE_CCSID_ERROR
(2111, X'83F') Source coded character set identifier not valid.

MQRC_SOURCE_INTEGER_ENC_ERROR
(2112, X'840') Source integer encoding not recognized.

MQRC_SOURCE_LENGTH_ERROR
(2143, X'85F') Source length parameter not valid.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_TARGET_BUFFER_ERROR
(2146, X'862') Target buffer parameter not valid.

MQRC_TARGET_CCSID_ERROR
(2115, X'843') Target coded character set identifier not valid.

MQRC_TARGET_INTEGER_ENC_ERROR
(2116, X'844') Target integer encoding not recognized.

MQRC_TARGET_LENGTH_ERROR
(2144, X'860') Target length parameter not valid.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For more information on these reason codes, see Chapter 4, "Return codes," on page 661.

C invocation

```
MQXCNCV (Hconn, Options, SourceCCSID, SourceLength, SourceBuffer,
        TargetCCSID, TargetLength, TargetBuffer, &DataLength,
        &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;          /* Connection handle */
MQLONG  Options;        /* Options that control the action of
                        MQXCNCV */
MQLONG  SourceCCSID;    /* Coded character set identifier of string
                        before conversion */
MQLONG  SourceLength;   /* Length of string before conversion */
MQCHAR  SourceBuffer[n]; /* String to be converted */
MQLONG  TargetCCSID;    /* Coded character set identifier of string
                        after conversion */
MQLONG  TargetLength;   /* Length of output buffer */
MQCHAR  TargetBuffer[n]; /* String after conversion */
MQLONG  DataLength;     /* Length of output string */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

COBOL invocation (i5/OS only)

```
CALL 'MQXCNCV' USING HCONN, OPTIONS, SOURCECCSID, SOURCELENGTH,  
                    SOURCEBUFFER, TARGETCCSID, TARGETLENGTH,  
                    TARGETBUFFER, DATALENGTH, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle  
01 HCONN          PIC S9(9) BINARY.  
** Options that control the action of MQXCNCV  
01 OPTIONS        PIC S9(9) BINARY.  
** Coded character set identifier of string before conversion  
01 SOURCECCSID    PIC S9(9) BINARY.  
** Length of string before conversion  
01 SOURCELENGTH   PIC S9(9) BINARY.  
** String to be converted  
01 SOURCEBUFFER    PIC X(n).  
** Coded character set identifier of string after conversion  
01 TARGETCCSID    PIC S9(9) BINARY.  
** Length of output buffer  
01 TARGETLENGTH   PIC S9(9) BINARY.  
** String after conversion  
01 TARGETBUFFER    PIC X(n).  
** Length of output string  
01 DATALENGTH    PIC S9(9) BINARY.  
** Completion code  
01 COMPCODE        PIC S9(9) BINARY.  
** Reason code qualifying COMPCODE  
01 REASON          PIC S9(9) BINARY.
```

System/390 assembler invocation

```
CALL MQXCNCV, (HCONN, OPTIONS, SOURCECCSID, SOURCELENGTH,      X  
              SOURCEBUFFER, TARGETCCSID, TARGETLENGTH, TARGETBUFFER, X  
              DATALENGTH, COMPCODE, REASON)
```

Declare the parameters as follows:

```
HCONN          DS F      Connection handle  
OPTIONS        DS F      Options that control the action of MQXCNCV  
SOURCECCSID    DS F      Coded character set identifier of string before  
* conversion  
SOURCELENGTH   DS F      Length of string before conversion  
SOURCEBUFFER   DS CL(n)  String to be converted  
TARGETCCSID    DS F      Coded character set identifier of string after  
* conversion  
TARGETLENGTH   DS F      Length of output buffer  
TARGETBUFFER   DS CL(n)  String after conversion  
DATALENGTH     DS F      Length of output string  
COMPCODE       DS F      Completion code  
REASON         DS F      Reason code qualifying COMPCODE
```

MQ_DATA_CONV_EXIT – Data conversion exit

This call definition describes the parameters that are passed to the data-conversion exit.

No entry point called MQ_DATA_CONV_EXIT is actually provided by the queue manager (see usage note 11 on page 701).

This definition is part of the MQSeries Data Conversion Interface (DCI), which is one of the MQSeries framework interfaces.

Syntax

`MQ_DATA_CONV_EXIT` (*DataConvExitParms*, *MsgDesc*, *InBufferLength*, *InBuffer*, *OutBufferLength*, *OutBuffer*)

Parameters

The `MQ_DATA_CONV_EXIT` call has the following parameters.

DataConvExitParms (MQDXP) – input/output

Data-conversion exit parameter block.

This structure contains information relating to the invocation of the exit. The exit sets information in this structure to indicate the outcome of the conversion. See “MQDXP – Data-conversion exit parameter” on page 684 for details of the fields in this structure.

MsgDesc (MQMD) – input/output

Message descriptor.

On input to the exit, this is the message descriptor associated with the message data passed to the exit in the *InBuffer* parameter.

Note: The *MsgDesc* parameter passed to the exit is always the most-recent version of MQMD supported by the queue manager which invokes the exit. If the exit is intended to be portable between different environments, the exit should check the *Version* field in *MsgDesc* to verify that the fields that the exit needs to access are present in the structure.

In the following environments, the exit is passed a version-2 MQMD: AIX, HP-UX, OS/2, i5/OS, Solaris, Linux, Windows. In all other environments that support the data conversion exit, the exit is passed a version-1 MQMD.

On output, the exit should change the *Encoding* and *CodedCharSetId* fields to the values requested by the application, if conversion was successful; these changes will be reflected back to the application. Any other changes that the exit makes to the structure are ignored; they are not reflected back to the application.

If the exit returns `MQXDR_OK` in the *ExitResponse* field of the MQDXP structure, but does not change the *Encoding* or *CodedCharSetId* fields in the message descriptor, the queue manager returns for those fields the values that the corresponding fields in the MQDXP structure had on input to the exit.

InBufferLength (MQLONG) – input

Length in bytes of *InBuffer*.

This is the length of the input buffer *InBuffer*, and specifies the number of bytes to be processed by the exit. *InBufferLength* is the lesser of the length of the message data prior to conversion, and the length of the buffer provided by the application on the MQGET call.

The value is always greater than zero.

InBuffer (MQBYTE×InBufferLength) – input

Buffer containing the unconverted message.

This contains the message data prior to conversion. If the exit is unable to convert the data, the queue manager returns the contents of this buffer to the application after the exit has completed.

Note: The exit should not alter *InBuffer*; if this parameter is altered, the results are undefined.

In the C programming language, this parameter is defined as a pointer-to-void.

OutBufferLength (MQLONG) – input

Length in bytes of *OutBuffer*.

This is the length of the output buffer *OutBuffer*, and is the same as the length of the buffer provided by the application on the MQGET call.

The value is always greater than zero.

OutBuffer (MQBYTE×OutBufferLength) – output

Buffer containing the converted message.

On output from the exit, if the conversion was successful (as indicated by the value MQXDR_OK in the *ExitResponse* field of the *DataConvExitParms* parameter), *OutBuffer* contains the message data to be delivered to the application, in the requested representation. If the conversion was unsuccessful, any changes that the exit has made to this buffer are ignored.

In the C programming language, this parameter is defined as a pointer-to-void.

Usage notes

1. A data-conversion exit is a user-written exit which receives control during the processing of an MQGET call. The function performed by the data-conversion exit is defined by the provider of the exit; however, the exit must conform to the rules described here, and in the associated parameter structure MQDXP. The programming languages that can be used for a data-conversion exit are determined by the environment.
2. The exit is invoked only if *all* of the following are true:
 - The MQGMO_CONVERT option is specified on the MQGET call
 - The *Format* field in the message descriptor is not MQFMT_NONE
 - The message is not already in the required representation; that is, one or both of the message's *CodedCharSetId* and *Encoding* is different from the value specified by the application in the message descriptor supplied on the MQGET call
 - The queue manager has not already done the conversion successfully
 - The length of the application's buffer is greater than zero
 - The length of the message data is greater than zero
 - The reason code so far during the MQGET operation is MQRC_NONE or MQRC_TRUNCATED_MSG_ACCEPTED
3. When an exit is being written, consideration should be given to coding the exit in a way that will allow it to convert messages that have been truncated. Truncated messages can arise in the following ways:

- The receiving application provides a buffer that is smaller than the message, but specifies the MQGMO_ACCEPT_TRUNCATED_MSG option on the MQGET call.

In this case, the *Reason* field in the *DataConvExitParms* parameter on input to the exit will have the value MQRC_TRUNCATED_MSG_ACCEPTED.

- The sender of the message truncated it before sending it. This can happen with report messages, for example (see “Conversion of report messages” on page 683 for more details).

In this case, the *Reason* field in the *DataConvExitParms* parameter on input to the exit will have the value MQRC_NONE (if the receiving application provided a buffer that was big enough for the message).

Thus the value of the *Reason* field on input to the exit cannot always be used to decide whether the message has been truncated.

The distinguishing characteristic of a truncated message is that the length provided to the exit in the *InBufferLength* parameter will be *less than* the length implied by the format name contained in the *Format* field in the message descriptor. The exit should therefore check the value of *InBufferLength* before attempting to convert any of the data; the exit *should not* assume that the full amount of data implied by the format name has been provided.

If the exit has *not* been written to convert truncated messages, and *InBufferLength* is less than the value expected, the exit should return MQXDR_CONVERSION_FAILED in the *ExitResponse* field of the *DataConvExitParms* parameter, with the *CompCode* and *Reason* fields set to MQCC_WARNING and MQRC_FORMAT_ERROR respectively.

If the exit *has* been written to convert truncated messages, the exit should convert as much of the data as possible (see next usage note), taking care not to attempt to examine or convert data beyond the end of *InBuffer*. If the conversion completes successfully, the exit should leave the *Reason* field in the *DataConvExitParms* parameter unchanged. This has the effect of returning MQRC_TRUNCATED_MSG_ACCEPTED if the message was truncated by the receiver’s queue manager, and MQRC_NONE if the message was truncated by the sender of the message.

It is also possible for a message to expand *during* conversion, to the point where it is bigger than *OutBuffer*. In this case the exit must decide whether to truncate the message; the *AppOptions* field in the *DataConvExitParms* parameter will indicate whether the receiving application specified the MQGMO_ACCEPT_TRUNCATED_MSG option.

4. Generally it is recommended that all of the data in the message provided to the exit in *InBuffer* is converted, or that none of it is. An exception to this, however, occurs if the message is truncated, either before conversion or during conversion; in this case there may be an incomplete item at the end of the buffer (for example: one byte of a double-byte character, or 3 bytes of a 4-byte integer). In this situation it is recommended that the incomplete item should be omitted, and unused bytes in *OutBuffer* set to nulls. However, complete elements or characters within an array or string *should* be converted.
5. When an exit is needed for the first time, the queue manager attempts to load an object that has the same name as the format (apart from extensions). The object loaded must contain the exit that processes messages with that format name. It is recommended that the exit name, and the name of the object that contain the exit, should be identical, although not all environments require this.

6. A new copy of the exit is loaded when an application attempts to retrieve the first message that uses that *Format* since the application connected to the queue manager. For CICS or IMS applications, this means when the CICS or IMS subsystem connected to the queue manager. A new copy may also be loaded at other times, if the queue manager has discarded a previously-loaded copy. For this reason, an exit should not attempt to use static storage to communicate information from one invocation of the exit to the next – the exit may be unloaded between the two invocations.
7. If there is a user-supplied exit with the same name as one of the built-in formats supported by the queue manager, the user-supplied exit does not replace the built-in conversion routine. The only circumstances in which such an exit is invoked are:
 - If the built-in conversion routine cannot handle conversions to or from either the *CodedCharSetId* or *Encoding* involved, or
 - If the built-in conversion routine has failed to convert the data (for example, because there is a field or character which cannot be converted).
8. The scope of the exit is environment-dependent. *Format* names should be chosen so as to minimize the risk of clashes with other formats. It is recommended that they start with characters that identify the application defining the format name.
9. The data-conversion exit runs in an environment similar to that of the program which issued the MQGET call; environment includes address space and user profile (where applicable). The program could be a message channel agent sending messages to a destination queue manager that does not support message conversion. The exit cannot compromise the queue manager's integrity, since it does not run in the queue manager's environment.
10. The only MQI call which can be used by the exit is MQXCNVC; attempting to use other MQI calls fails with reason code MQRC_CALL_IN_PROGRESS, or other unpredictable errors.
11. No entry point called MQ_DATA_CONV_EXIT is actually provided by the queue manager. However, a **typedef** is provided for the name MQ_DATA_CONV_EXIT in the C programming language, and this can be used to declare the user-written exit, to ensure that the parameters are correct. The name of the exit should be the same as the format name (the name contained in the *Format* field in MQMD), although this is not required in all environments.

The following example illustrates how the exit that processes the format MYFORMAT should be declared in the C programming language:

```
#include "cmqc.h"
#include "cmqxc.h"

MQ_DATA_CONV_EXIT MYFORMAT;

void MQENTRY MYFORMAT(
    PMQDXP  pDataConvExitParms, /* Data-conversion exit parameter
                                block */
    PMQMD   pMsgDesc,           /* Message descriptor */
    MQLONG  InBufferLength,     /* Length in bytes of InBuffer */
    PMQVOID pInBuffer,         /* Buffer containing the unconverted
                                message */
    MQLONG  OutBufferLength,    /* Length in bytes of OutBuffer */
    PMQVOID pOutBuffer)        /* Buffer containing the converted
                                message */
{
    /* C language statements to convert message */
}
```

12. On z/OS, if an API-crossing exit is also in force, it is called after the data-conversion exit.

C invocation

```
exitname (&DataConvExitParms, &MsgDesc, InBufferLength,
         InBuffer, OutBufferLength, OutBuffer);
```

The parameters passed to the exit are declared as follows:

```
MQDXP  DataConvExitParms; /* Data-conversion exit parameter block */
MQMD   MsgDesc;          /* Message descriptor */
MQLONG InBufferLength;   /* Length in bytes of InBuffer */
MQBYTE InBuffer[n];     /* Buffer containing the unconverted
                        message */
MQLONG OutBufferLength;  /* Length in bytes of OutBuffer */
MQBYTE OutBuffer[n];    /* Buffer containing the converted
                        message */
```

COBOL invocation (i5/OS only)

```
CALL 'exitname' USING DATACONVEXITPARMS, MSGDESC, INBUFFERLENGTH,
                     INBUFFER, OUTBUFFERLENGTH, OUTBUFFER.
```

The parameters passed to the exit are declared as follows:

```
** Data-conversion exit parameter block
01 DATACONVEXITPARMS.
   COPY CMQDXPV.
** Message descriptor
01 MSGDESC.
   COPY CMQMDV.
** Length in bytes of INBUFFER
01 INBUFFERLENGTH PIC S9(9) BINARY.
** Buffer containing the unconverted message
01 INBUFFER PIC X(n).
** Length in bytes of OUTBUFFER
01 OUTBUFFERLENGTH PIC S9(9) BINARY.
** Buffer containing the converted message
01 OUTBUFFER PIC X(n).
```

System/390 assembler invocation

```
CALL EXITNAME,(DATACONVEXITPARMS,MSGDESC,INBUFFERLENGTH, X
               INBUFFER,OUTBUFFERLENGTH,OUTBUFFER)
```

The parameters passed to the exit are declared as follows:

```
DATACONVEXITPARMS CMQDXPA , Data-conversion exit parameter block
MSGDESC           CMQMDA , Message descriptor
INBUFFERLENGTH   DS      F Length in bytes of INBUFFER
INBUFFER         DS      CL(n) Buffer containing the unconverted
*               message
OUTBUFFERLENGTH  DS      F Length in bytes of OUTBUFFER
OUTBUFFER        DS      CL(n) Buffer containing the converted
*               message
```

End of product-sensitive programming interface

Chapter 9. Properties specified as MQRFH2 elements

Non-message descriptor properties can be specified as elements in MQRFH2 header folders. Overview of MQRFH2 elements being specified as properties.

This retains compatibility with the previous versions of the WebSphere MQ JMS and XMS clients. This section describes how to specify properties in MQRFH2 headers.

To use MQRFH2 elements as properties, specify the elements as described in *Using Java*. This information supplements the information described in “MQRFH2 – Rules and formatting header 2” on page 308.

Mapping property data types to MQRFH2 data types

Message property types map to the following supported MQRFH2 data types. Table of how message property types map to supported MQRFH2 data types.

Any element without a data type is assumed to be of type “string”.

Message property type	MQRFH2 data type
MQBYTE[]	bin.hex
MQBOOL	boolean
MQINT8	i1
MQINT16	i2
MQINT32	i4
MQINT64	i8
MQFLOAT32	r4
MQFLOAT64	r8
MQCHAR[]	string

An MQRFH2 data type of int, meaning an integer of unspecified size, is treated as if it were an i8.

A null value is indicated by the element attribute `xsi:nil='true'` as described in *Using Java*. Do not use the attribute `xsi:nil='false'` for non-null values.

For example, the following property has a null value:

```
<NullProperty xsi:nil='true'></NullProperty>
```

A byte or character string property can have an empty value. This is represented by an MQRFH2 element with a zero length element value.

For example, the following property has an empty value:

```
<EmptyProperty></EmptyProperty>
```

Supported MQRFH2 folders

Overview of the use of message descriptor fields as properties

In addition to the folders <jms>, <mcd>, <mqext>, and <usr> that are described in *Using Java*, WebSphere MQ supports the following additional folders:

- <mq>
This is used and reserved for MQ-defined properties that are used by WebSphere MQ.
- <mq_usr>
This can be used to transport any application-defined properties that are not exposed as JMS user-defined properties, as the properties might not meet the requirements of a JMS property. Also, this folder can contain groups that the <usr> folder cannot.
- Any folder marked with the content='properties' attribute.
Such a folder is equivalent to the <mq_usr> folder in content.

WebSphere MQ also supports the following folders that are already in use by WAS/SIB:

- <sib>
This is used and reserved for WAS/SIB system message properties that are not exposed as JMS properties, or are mapped to JMS_IBM_* properties, but are exposed to WAS/SIB applications; these include forward and reverse routing paths properties.
At least some cannot be exposed as JMS properties because they are byte arrays. Your application should not add properties to this folder; if it does, the value is either ignored or removed.
- <sib_usr>
This is used and reserved for WAS/SIB user message properties that cannot be exposed as JMS user properties because they are not of supported types; they are exposed to WAS/SIB applications.
These are user properties, that you can get or set through the SIMessage interface, but the content of the byte array is mapped to the required property value.
If your WebSphere MQ application writes an arbitrary bin.hex element to the folder, the application probably receives an IOException, as it is not of the format expected to restore. If you add anything other than a bin.hex element you receive a ClassCastException.
You should not attempt to make properties available to WAS/SIB by using this folder; instead the <usr> folder should be used for that purpose.
- <sib_context>
This is used for WAS/SIB system message properties that are not exposed to WAS/SIB user applications or as JMS properties. These include security and transactional properties that are used for Web services and similar.
Your application should not add properties to this folder.
- <mqema>
This folder was used by WAS/SIB instead of the <mqext> folder.

Note that MQRFH2 folder names are case sensitive.

The following folders are reserved, in any mixture of lower or upper case characters:

- Any folder prefixed by mq or wmq; reserved for use by WebSphere MQ
- Any folder prefixed by sib; reserved for use by WAS/SIB.
- <Root> and <Body> folders; reserved but not used.

The following folders are *not* recognized as containing message properties:

- <psc>
Used by WebSphere Message Brokers to convey publish/subscribe command messages to the broker.
- <pscr>
Used by WebSphere Message Brokers to contain information from the broker, in response to publish/subscribe command messages.
- Any folder not defined by IBM, that is not marked with the `content='properties'` attribute.

Do not specify `content='properties'` on the <psc> or <pscr> folders. If you do so, these folders are treated as properties and WebSphere Message Brokers is likely to stop functioning as expected.

If your application is building messages with properties, in MQRFH2 headers to be recognized as an MQRFH2 header containing properties, the header must be in the list of headers that can be chained at the head of the message.

The MQRFH2 can be preceded by any number of “MQH” standard headers, or an MQCIH, an MQDLH, an MQIIH, an MQTM, an MQTMC2, or an MQXQH. Note that, for example, a string or an MQCFH ends parsing because they can not be chained.

It is possible for a message to contain multiple MQRFH2 headers all carrying message properties. Folders with the same name can coexist in different headers unless otherwise restricted, for example by WAS/SIB. The folders are treated as one logical folder, provided that they are all in significant headers.

While folders from the significant headers cannot be merged with those in nonsignificant headers, folders with the same name within the significant headers can be merged, removing any conflicting properties. Your applications must not depend on the layout of properties within their message.

MQRFH2 groups are parsed for properties in user-defined folders, that is, not the <wmq>, <jms>, <mcd>, <usr>, <mqext>, <sib>, <sib_usr>, <sib_context>, and <mqema> folders.

Groups in the IBM-defined folders, except for the <wmq_usr> folder, are not parsed for properties.

Mixed content is not allowed in an MQRFH2 folder,, a folder or group can contain either groups or properties, or a value, but not both.

A segment of a message, either the first or a subsequent segment, cannot contain WebSphere MQ-defined properties other than those in the message descriptor; so putting a message containing such properties with either MQMF_SEGMENT or MQMF_SEGMENTATION_ALLOWED set causes the put to fail with MQRC_SEGMENTATION_NOT_ALLOWED.

Note that WebSphere MQ-defined properties are, however, allowed in message groups.

Generation of MQRFH2 headers

If a queue manager adds one or more properties to a message or an application adds properties to a message using the MQSETMP call, then when a second application gets the message, WebSphere MQ converts the properties to their MQRFH2 representation and includes them in an existing MQRFH2, or into a new MQRFH2 which it generates.

If the message payload contains an MQRFH2 structure directly following the MQMD structure, an MQRFH, or an MQXQH, the MQRFH2 contains at least one message property, and the MQRFH2 has a NameValueCCSID value matching the CCSID of the new properties, then the new properties will be merged into the existing MQRFH2. If the message payload does not contain an MQRFH2, or it does not fulfil these criteria, then a new MQRFH2 will be generated and inserted in the payload (after the MQXQH or MQRFH if present) containing the MQRFH2 representation of the properties to be converted.

If the properties are merged into an existing MQRFH2, then the existing MQRFH2 will be searched for the folders which need to be merged. If a folder does not exist it will be added to the end of the existing folders. If the folder does exist then the folder will be searched: any matching existing properties will be overwritten and any new ones will be added at the end of the folder.

MQRFH2 folder restrictions

Overview of folder restrictions in MQRFH2 headers

The MQRFH2 restrictions apply to the following folders:

- Element names in the <usr> folder must not begin with the prefix JMS; such property names are reserved for use by JMS and are not valid for user-defined properties.

Such an element name does not cause parsing of the MQRFH2 to fail, but is not accessible to the WebSphere MQ message property APIs.

- Element names in the <usr> folder must not be, in any mixture of lower or uppercase, "NULL", "TRUE", "FALSE", "NOT", "AND", "OR", "BETWEEN", "LIKE", "IN", "IS" and "ESCAPE". These names match SQL keywords and make parsing selectors harder, because a property in the <usr> is the default folder used when no folder is specified for a given property in a selector.

Such an element name does not cause parsing of the MQRFH2 to fail, but is not accessible to the WebSphere MQ message property APIs.

- Element names in any folder considered to contain message properties must not contain the "." character (Unicode character U+002E), because this is used in property names to indicate the hierarchy.

Such an element name does not cause parsing of the MQRFH2 to fail, but is not accessible to the WebSphere MQ message property APIs.

In general, MQRFH2 headers that contain valid XML-style data can be parsed by WebSphere MQ without failure, although certain elements of the MQRFH2 are not accessible through the WebSphere MQ message property APIs.

MQRFH2 element name conflicts

Only one value can be attached to a message property, so, if an attempt to access a property leads to a conflict of values one is chosen in preference over another. Overview of conflicts within MQRFH2 element names.

The WebSphere MQ syntax for accessing MQRFH2 elements allows unique identification of an element, provided that a folder contains no elements with the same name. If a folder does contain more than one element with the same name, the value of the property used is the one closest to the head of the message.

This applies if two or more folders of the same name are contained in different significant MQRFH2 headers within the same message.

Given that a non-message descriptor property can be set either directly in the raw MQRFH2 header, or through “MQSETMP – Set message property” on page 556, this could result in a conflict when the MQGET call is processed where both contain the same property.

Should this happen, the property associated with the message by an API call takes preference over one in the message data, that is, the one in the raw MQRFH2 header. If a conflict results, it is considered to come logically before the message data.

Mapping from property names to MQRFH2 folder and element names

When using any of the APIs defined that ultimately generate MQRFH2 headers, in order to specify message properties (for example, MQ JMS), the property name is not necessarily the element name in the MQRFH2 folder. Overview of the differences between property names and element names in the MQRFH2 header.

Therefore, a mapping has to take place from the property name to the MQRFH2 element, and in the reverse way, taking into account both the folder name that contains the element as well as the element name. Some examples from WebSphere MQ JMS are already documented in *Using Java*.

Property name	MQRFH2 folder name	MQRFH2 element name
JMSDestination	jms	Dst
JMSType	mcd	Type, Set, Fmt
xxx (user defined, where xxx does not begin with JMS)	usr	xxx

Therefore, when a JMS application accesses the “JMSDestination” property this maps to the Dst element in the <jms> folder.

When specifying properties as MQRFH2 elements, WebSphere MQ defines its elements as follows:

Property name	MQRFH2 folder name	MQRFH2 group name	MQRFH2 element name
<Property>	<usr>	n/a	<Property>
<folder>,<Property>	<folder>	n/a	<Property>
<folder>, <group>,<Property>	<folder>	<group>	<Property>

For example, when a WebSphere MQ application attempts to access the Property1 property, this maps to the Property1 element in the <usr> folder. The wmq.Property2 property maps to the Property2 property in the <wmq> folder.

As Root and Body are reserved folder names, you cannot use Root or Body as your folder prefix; for example, Root.Property1 does not access a valid property.

If the property name contains more than one "." character, the MQRFH2 element name used is the one following the final "." character, and MQRFH2 groups are used to form a hierarchy; nested MQRFH2 groups are allowed.

The JMS header and provider-specific properties that are contained in an MQRFH2 in the <mcd>, <jms>, and <mqext> folders are accessed by a WebSphere MQ application using the short names defined in *Using Java*.

JMS user-defined properties are accessed from the <usr> folder. A WebSphere MQ application can use the <usr> folder for its application properties if it is acceptable for the property to appear to JMS applications as one of its user-defined properties.

If it is not acceptable, choose another folder; the <wmq_usr> folder is provided as a standard location for such non-JMS properties.

Your applications can specify and use any MQRFH2 folder with a well-defined use, not documented in Chapter 9, "Properties specified as MQRFH2 elements," on page 703 as long as you note the following:

1. The folder might already be in use, or might be used in the future, by another application providing undefined access to properties contained inside it; see *Property names* for the suggested naming convention for property names.
2. The properties are not accessible to previous versions of the Websphere MQ JMS or XMS client that can only access the <usr> folder for user defined properties
3. The folder must be marked with the attribute content with the value set to properties, for example, content='properties'.
"MQSETMP – Set message property" on page 556 automatically adds this attribute as required. This attribute must not be added to any of the IBM-defined folders, for example, <jms> and <usr>. Doing so, causes the message to be rejected by the WebSphere MQ JMS client prior to Version 7.0. with a MessageFormatException.

As the <usr> folder is the default location for properties of the <Property> syntax, this enables a WebSphere MQ application and a JMS application to access the same user-defined property value using the same name.

Mapping property descriptor fields into MQRFH2 headers

When a property is translated into an MQRFH2 element the following element attributes are used to specify the significant fields of the property descriptor: This describes how MQPD fields are translated to MQRFH2 element attributes.

Support

The Support property descriptor field is split into three element attributes

- The sr element attribute specifies values in the MQPD_REJECT_UNSUP_MASK bit mask.

- The sa element attribute specifies values in the MQPD_ACCEPT_UNSUP_MASK bit mask.
- The sx element attribute specifies values in the MQPD_ACCEPT_UNSUP_IF_XMIT_MASK bit mask.

These element attributes are only valid in the <mq> folder and are ignored if set on elements in the other folders containing properties.

Table 103.

Support value	MQRFH2 element attribute	MQRFH2 attribute value
MQPD_SUPPORT_OPTIONAL	sa	optional This is the default value.
MQPD_SUPPORT_REQUIRED	sr	required
MQPD_SUPPORT_REQUIRED_IF_LOCAL	sx	local

Context

Use the context element attribute to indicate the message context to which a property belongs. Use one value only. This element attribute is valid on a property in any folder containing properties.

Table 104.

Context value	MQRFH2 attribute value
MQPD_NO_CONTEXT	none This is the default value.
MQPD_USER_CONTEXT	user

CopyOptions

Use the copy element attribute to indicate messages into which a property should be copied. More than one value is acceptable; separate multiple values with a comma. For example copy='reply' and copy='publish,report' are both valid. This element attribute is valid on a property in any folder containing properties.

Note: In the attribute definition, a single apostrophe or double quotes are valid use, for example copy='reply' or copy="report"

Table 105.

CopyOption value	MQRFH2 attribute value
MQPD_COPY_FORWARD	forward
MQPD_COPY_REPLY	reply
MQPD_COPY_REPORT	report
MQPD_COPY_PUBLISH	publish
MQPD_COPY_ALL	all Do not specify this with any other value. When used with another value, this takes precedence over any value except none.

Table 105. (continued)

CopyOption value	MQRFH2 attribute value
MQPD_COPY_DEFAULT	<p>default</p> <p>This is the default value. It is equivalent to specifying the three values MQCOPY_FORWARD, MQCOPY_REPORT and MQCOPY_PUBLISH.</p> <p>Do not specify this with any other value.</p>
MQPD_COPY_NONE	<p>none</p> <p>Do not specify this with any other value. When used with another value, this takes precedence.</p>

Restrictions to the <mq> MQRFH2 folder

When a message is put on to a queue, it is searched for an <mq> folder so that the message can be processed according to its MQ-defined properties. To allow the efficient parsing of MQ-defined properties, the following restrictions apply to the folder:

- Only properties in the first significant <mq> folder in the message are acted upon by MQ; properties in any other <mq> folder in the message are ignored.
- If the folder is in UTF-8, only single-byte UTF-8 characters are allowed in the folder. A multi-byte character in the folder, can cause parsing to fail, and the message to be rejected.
- Do not include MQRFH2 groups in the <mq> folder. The presence of Unicode character U+003C in a property value will cause the message to be rejected.
- Do not use escape strings in the folder. An escape string is treated as the actual value of the element.
- Only Unicode character U+0020 is treated as white space within the folder. All other characters are treated as significant and can cause parsing of the folder to fail, and the message to be rejected.

If parsing of the <mq> folder fails, or if the folder does not observe these restrictions, the message is rejected with CompCode MQCC_FAILED and Reason MQRC_RFH_RESTRICTED_FORMAT_ERR.

MQRFH2 headers that are not valid

At the time an MQPUT, MQPUT1, or MQGET call processes, a partial parsing of any MQRFH2 headers in the message can occur to check what folders are included, and to determine if the folders contain properties. Overview of MQRFH2 headers that are not valid.

If the partial parsing of the message cannot complete successfully because the structure is not valid, for example, the StrucLength field is too small, then:

- The MQPUT or MQPUT1 call fails with reason code MQRC_RFH_ERROR, if it can be determined that the application includes some WebSphere MQ Version 7 option, so that existing applications do not fail.
- The MQGET call returns successfully, and the MQRFH2 containing the error is returned in the buffer you provided.

| If the partial parsing fails because it cannot be detected whether a particular folder
| contains properties or not, for example, the folder begins <<jms, so parsing fails
| before the folder name is determined, then:

- The MQPUT or MQPUT1 call fails with reason code MQRC_RFH_FORMAT_ERROR, if it can be determined that the application includes some WebSphere MQ Version 7 option, so that existing applications do not fail.
- The MQGET call returns successfully, and the MQRFH2 containing the error is returned in the buffer you provided.
- While internally within the queue manager, the message is not rejected due to the badly formatted folder, but the folder is always treated as if no properties were contained inside it.

| A message can flow through the queue manager network with a folder containing
| such a syntax error, but never being parsed and detected, while one or more
| folders in the message are:

- Valid
- Successfully parsed
- Used in the processing of the message

| Therefore, detection is not guaranteed.

| If one of your applications uses “MQSETMP – Set message property” on page 556,
| or MQINQMP to access a property, and in so doing this causes an MQRFH2 folder
| to be fully parsed, detecting an error such that parsing cannot complete, this is
| indicated by an appropriate return code to the API call. No properties in the folder
| are made available to the application.

| If an attempt is made to fully parse an MQRFH2 folder and the parser finds
| unrecognized element attributes, or an unrecognized data type, parsing continues
| and complete successfully with no warnings being issued; this does not constitute
| a parsing error.

Chapter 10. Code page conversion

Each national language section lists the following information:

- The native CCSIDs supported
- The code page conversions that are **not** supported

The following terms are used in the information:

-8 Indicates for HP-UX that the CCSID is for the HP-UX defined codeset *roman8*

AIX Indicates WebSphere MQ for AIX

OVMS

Indicates WebSphere MQ for HP OpenVMS

HP-UX

Indicates WebSphere MQ for HP-UX

Linux Indicates WebSphere MQ for Linux for Intel and WebSphere MQ for Linux for zSeries

NSS Indicates WebSphere MQ for HP NonStop Server

OS/400

Indicates WebSphere MQ for i5/OS

Solaris

Indicates WebSphere MQ for Solaris

Tru64 Indicates MQSeries for Compaq Tru64 UNIX

Windows

Indicates WebSphere MQ for Windows

z/OS Indicates WebSphere MQ for z/OS

The default for data conversion is for the conversion to be performed at the target (receiving) system.

If the source product supports the conversion a channel can be set up and data exchanged by setting the channel attribute **DataConversion** to YES at the source.

Note:

1. Conversion for WebSphere MQ client information takes place in the server, so the server must support conversion from the client CCSID to the server CCSID.
2. The conversion may include support added by CSD/PTF to the latest version of WebSphere MQ. Check the content of the latest service level to see if you need to install a CSD/PTF to enable this conversion.

See Table 106 on page 714 for a cross reference between some of the CCSID numbers and some industry codeset names.

Codeset names and CCSIDs

WebSphere MQ for z/OS provides more conversion than is listed in the language specific tables.

Table 106. Codeset names and CCSIDs

Codeset names	CCSIDs
ISO 8859-1	819
ISO 8859-2	912
ISO 8859-3	913
ISO 8859-5	915
ISO 8859-6	1089
ISO 8859-7	813
ISO 8859-8	916
ISO 8859-9	920
ISO 8859-13	921
ISO 8859-15 (euro)	923
big5	950
eucJP	954 5050 33722
eucKR	970
eucTW	964
eucCN	1383
PCK	943
GBK	1386
koi8-r	878

A complete list of conversions provided is shown in Table 107 on page 739.

National languages

The languages supported by WebSphere MQ are:

- US English – see topic “US English” on page 715
- German – see topic “German” on page 716
- Danish and Norwegian – see topic “Danish and Norwegian” on page 716
- Finnish and Swedish – see topic “Finnish and Swedish” on page 717
- Italian – see topic “Italian” on page 718
- Spanish – see topic “Spanish” on page 719
- UK English / Gaelic – see topic “UK English /Gaelic” on page 719
- French – see topic “French” on page 720
- Multilingual – see topic “Multilingual” on page 720
- Portuguese – see topic “Portuguese” on page 721
- Icelandic – see topic “Icelandic” on page 721
- Eastern European languages – see topic “Eastern European languages” on page 722
- Cyrillic – see topic “Cyrillic” on page 723
- Estonian – see topic “Estonian” on page 724
- Latvian and Lithuanian – see topic “Latvian and Lithuanian” on page 725
- Ukranian – see topic “Ukrainian” on page 726
- Greek – see topic “Greek” on page 726

- Turkish – see topic “Turkish” on page 727
- Hebrew – see topic “Hebrew” on page 727
- Farsi – see topic “Farsi” on page 729
- Urdu – see topic “Urdu” on page 729
- Thai – see topic “Thai” on page 730
- Lao – see topic “Lao” on page 730
- Vietnamese – see topic “Vietnamese” on page 730
- Japanese Latin SBCS – see topic “Japanese Latin SBCS” on page 731
- Japanese Katakana SBCS – see topic “Japanese Katakana SBCS” on page 732
- Japanese Kanji/ Latin Mixed – see topic “Japanese Kanji/ Latin Mixed” on page 734
- Japanese Kanji/ Katakana Mixed – see topic “Japanese Kanji/ Katakana Mixed” on page 735
- Korean – see topic “Korean” on page 737
- Simplified Chinese – see topic “Simplified Chinese” on page 737
- Traditional Chinese – see topic “Traditional Chinese” on page 738

US English

Details of CCSIDs and CCSID conversion for US English.

The following table shows the native CCSIDs for US English on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	37, 924, 1140
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	437, 850, 1252, 5348, 858
OVMS, NSS, Solaris, Linux	819, 923
Tru64	819, 850, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

i5/OS

Code page:

37 Does not convert to code pages 923, 858

924 Does not convert to code pages 437, 858, 1051, 1140, 1252, 1275, 5348

1140 Does not convert to code pages 924, 1051, 1275

i5/OS

Code page:

37 Does not convert to code pages 923, 858, 5348

924 Does not convert to code pages 437, 819, 850, 858, 1051, 1140, 1252, 1275, 5348

1140 Does not convert to code pages 924, 1051, 1275, 5348

DEC-OVMS, SINIX, DC/OSx

Code page:

819 Does not convert to code pages 1252, 1275

NCR

Code page:

819 Does not convert to code pages 1252, 1275

437 Does not convert to code pages 1252, 1275

850 Does not convert to code pages 1252, 1275

German

Details of CCSIDs and CCSID conversion for German

The following table shows the native CCSIDs for German on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	273, 924, 1141
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	437, 850, 858, 1252, 5348
OVMS, NSS, Solaris, Linux, Tru64	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

i5/OS

Code page:

273 Does not convert to code pages 858, 923, 924, 1275

924 Does not convert to code pages 273, 437, 858, 1051, 1141, 1252, 1275, 5348

1141 Does not convert to code pages 924, 1051, 1275

Danish and Norwegian

Details of CCSIDs and CCSID conversion for Danish and Norwegian

The following table shows the native CCSIDs for Danish and Norwegian on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	277, 924, 1142
AIX	819, 923, 5348

Platform	Native CCSIDs
HP-UX	819, 923, 1051
Windows	850, 858, 865, 1252, 5348
OVMS, NSS, Solaris, Linux, Tru64	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

i5/OS

Code page:

277 Does not convert to code pages 858, 923, 924, 1275

924 Does not convert to code pages 277, 858, 865, 1051, 1142, 1252, 1275, 5348

1142 Does not convert to code pages 924, 865, 1051, 1275

AIX

Code page:

819 Does not convert to code page 865

HP-UX

Code page:

1051 Does not convert to code page 865

Windows

Code page:

865 Does not convert to code pages 1051, 1275

Finnish and Swedish

Details of CCSIDs and CCSID conversion for Finnish and Swedish

The following table shows the native CCSIDs for Finnish and Swedish on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	278, 924, 1143
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	437, 850, 858, 865, 1252, 5348
OVMS, NSS, Solaris, Linux, Tru64	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

i5/OS

Code page:

- 278 Does not convert to code pages 858, 923, 924, 1275
- 924 Does not convert to code pages 278, 437, 858, 865, 1051, 1143, 1252, 1275, 5348
- 1143 Does not convert to code pages 865, 924, 1051, 1275

AIX

Code page:

- 819 Does not convert to code page 865
- 850 Does not convert to code page 865

HP-UX

Code page:

- 1051 Does not convert to code page 865

Windows

Code page:

- 865 Does not convert to code pages 1051, 1275

Italian

Details of CCSIDs and CCSID conversion for Italian

The following table shows the native CCSIDs for Italian on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	280, 924, 1144
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	437, 850, 858, 1252, 5348
OVMS, NSS, Solaris, Linux, Tru64	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

i5/OS

Code page:

- 280 Does not convert to code pages 858, 923, 924, 1275
- 924 Does not convert to code pages 280, 437, 858, 1051, 1144, 1252, 1275, 5348
- 1144 Does not convert to code pages 924, 1051, 1275

Spanish

Details of CCSIDs and CCSID conversion for Spanish

The following table shows the native CCSIDs for Spanish on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	284, 924, 1145
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	437, 850, 858, 1252, 5348
OVMS, NSS, Solaris, Linux, Tru64	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

i5/OS

Code page:

284 Does not convert to code pages 858, 923, 924, 1275

924 Does not convert to code pages 284, 437, 858, 1051, 1145, 1252, 1275, 5348

1145 Does not convert to code pages 924, 1051, 1275

UK English /Gaelic

Details of CCSIDs and CCSID conversion for UK English/Gaelic

The following table shows the native CCSIDs for UK English / Gaelic on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	285, 924, 1146
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	437, 850, 858, 1252, 5348
OVMS, NSS, Solaris, Linux, Tru64	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

i5/OS

Code page:

285 Does not convert to code pages 858, 923, 924, 1275

924 Does not convert to code pages 285, 437, 858, 1051, 1146, 1252, 1275, 5348

1146 Does not convert to code pages 924, 1051, 1275

French

Details of CCSIDs and CCSID conversion for French

The following table shows the native CCSIDs for French on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	297, 924, 1147
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	437, 850, 858, 1252, 5348
OVMS, NSS, Solaris, Linux, Tru64	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

i5/OS

Code page:

297 Does not convert to code pages 858, 923, 924, 1275, 5348

924 Does not convert to code pages 297, 437, 858, 1051, 1147, 1252, 1275, 5348

1147 Does not convert to code pages 924, 1051, 1275

Multilingual

Details of CCSIDs and CCSID conversion for Multilingual

The following table shows the native CCSIDs for multilingual conversion on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	500, 924, 1148
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	437, 850, 858, 1252, 5348
OVMS, NSS, SINIX, Solaris, Linux, Tru64	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

i5/OS

Code page:

500 Does not convert to code pages 858, 923

924 Does not convert to code pages 437, 858, 1051, 1148, 1252, 1275, 5348

1148 Does not convert to code pages 924, 1051, 1275

Portuguese

Details of CCSIDs and CCSID conversion for Portuguese

The following table shows the native CCSIDs for Portuguese on supported platforms:

Platform	Native CCSIDs
i5/OS	37, 500, 924, 1140
z/OS	500, 924, 1140
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	850, 858, 860, 1252, 5348
OVMS, NSS, Solaris, Linux, Tru64	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

i5/OS

Code page:

37 Does not convert to code pages 858, 923, 1275

500 Does not convert to code pages 858, 923, 1275

924 Does not convert to code pages 858, 860, 1051, 1140, 1252, 1275, 5348

1140 Does not convert to code pages 860, 924, 1051, 1275

HP-UX

Code page:

1051 Does not convert to code page 860

Windows

Code page:

860 Does not convert to code pages 1051, 1275

Icelandic

Details of CCSIDs and CCSID conversion for Icelandic

The following table shows the native CCSIDs for Icelandic on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	871, 924, 1149
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	850, 858, 861, 1252, 5348
OVMS, NSS, Solaris, Linux, Tru64	819, 923

Platform	Native CCSIDs
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

i5/OS

Code page:

871 Does not convert to code pages 858, 923, 924, 1275, 5348

924 Does not convert to code pages 858, 861, 871, 1051, 1149, 1252, 1275, 5348

1149 Does not convert to code pages 924, 1051, 1275

HP-UX

Code page:

1051 Does not convert to code page 861

Windows

Code page:

861 Does not convert to code pages 1051, 1275

Eastern European languages

The typical languages using these CCSIDs include Albanian, Croatian, Czech, Hungarian, Polish, Romanian, Serbian, Slovak, and Slovenian. Details of CCSIDs and CCSID conversion for Eastern European Languages

The following table shows the native CCSIDs for Eastern European languages on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	870, 1153
Windows	852, 1250, 5346, 9044
AIX, OVMS, HP-UX, NSS, Solaris, Linux, Tru64	912
Eastern European Apple client	1282
Romanian Apple client	1285
Croatian Apple client	1284

z/OS

Code page:

870 Does not convert to code pages 1284, 1285

1153 Does not convert to code pages 1250, 1284, 1285

i5/OS

Code page:

- 870 Does not convert to code pages 1284, 1285, 5346, 9044
- 1153 Does not convert to code pages 1282, 1284, 1285, 5346, 9044

HP-UX, Solaris, Linux

Code page:

- 912 Does not convert to code pages 1284, 1285

OVMS, NSS

Code page:

- 912 Does not convert to code pages 1153, 1284, 1285, 9044

Windows

Code page:

- 852 Does not convert to code pages 1284, 1285
- 1250 Does not convert to code pages 1284, 1285
- 9044 Does not convert to code pages 912, 1282, 1284, 1285

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

Cyrillic

The typical languages using these CCSIDs include Belarussian, Bulgarian, Macedonian, Russian, and Serbian. Details of CCSIDs and CCSID conversion for Cyrillic

The following table shows the native CCSIDs for Cyrillic on supported platforms:

Platform	Native CCSIDs
z/OS	1025
i5/OS	880, 1025
Windows	855, 866, 1131, 1251, 5347
Solaris	878, 915
AIX, OVMS, HP-UX, Linux, NSS, Tru64	915
Apple client	1283

i5/OS

Code page:

- 880 Does not convert to code pages 855, 866, 878, 1131, 5347
- 1025 Does not convert to code pages 878, 5347

Windows

Code page:

- 855 Does not convert to code page 1131

- 866 Does not convert to code page 1131
- 1131 Does not convert to code pages 855, 866, 880, 1283

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

Estonian

Details of CCSIDs and CCSID conversion for Estonian

The following table shows the native CCSIDs for Estonian on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	1122, 1157
Windows	902, 922, 1257, 5353, 9449
AIX, HP-UX, Solaris, Linux	902, 922
OVMS, NSS, Tru64	922

z/OS

Code page:

- 1122 Does not convert to code pages 902, 1157, 9449
- 1157 Does not convert to code pages 922, 1122, 1257, 9449

i5/OS

Code page:

- 1122 Does not convert to code pages 902, 5353, 9449
- 1157 Does not convert to code pages 922, 5353, 9449

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

HP-UX, Solaris, Linux

Code page:

- 902 Does not convert to code pages 922, 1122, 9449
- 922 Does not convert to code pages 902, 1157, 9449

Windows

Code page:

- 5353 Does not convert to code page 9449
- 9449 Does not convert to code pages 902, 922, 1122, 1157, 1257, 5353
- 902 Does not convert to code pages 922, 1122, 9449

OVMS, NSS, Tru64

Code page:

922 Does not convert to code pages 902, 1157, 9449

Latvian and Lithuanian

Details of CCSIDs and CCSID conversion for Latvian and Lithuanian

The following table shows the native CCSIDs for Latvian and Lithuanian on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	1112, 1156
Windows	901, 921, 1257, 5353, 9449
AIX, HP-UX, Solaris, Linux	901, 921
OVMS, NSS, Tru64	921

z/OS

Code page:

1112 Does not convert to code pages 901, 1156, 9449

1156 Does not convert to code pages 901, 1156, 9449

i5/OS

Code page:

1112 Does not convert to code page 5353

1153 Does not convert to code pages 921, 5353, 9449

HP-UX, Solaris, Linux

Code page:

902 Does not convert to code pages 921, 1112, 1257, 9449

921 Does not convert to code pages 901, 1156, 9449

Windows

Code page:

901 Does not convert to code pages 921, 1112, 1257, 9449

5355 Does not convert to code page 9449

9449 Does not convert to code pages 901, 921, 1112, 1156, 1257

OVMS, NSS, Tru64

Code page:

921 Does not convert to code pages 901, 1156, 9449

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

Ukrainian

Details of CCSIDs and CCSID conversion for Ukrainian

The following table shows the native CCSIDs for Ukrainian on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	1123
Windows	1124, 1125, 1251, 5347
AIX, OVMS, HP-UX, NSS, Solaris, Linux, Tru64	1124

i5/OS

Code page:

1123 Does not convert to code page 5347

HP-UX

Code page:

1124 Does not convert to code page 5347

Windows

Code page:

1125 Does not convert to code page 1123

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

Greek

Details of CCSIDs and CCSID conversion for Greek

The following table shows the native CCSIDs for Greek on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	875
HP-UX	813 (see note)
Windows	869, 1253, 5349
AIX, OVMS, NCR, NSS, Solaris, Linux, Tru64	813
Apple client	1280
DOS client	737

Note: Only the ISO codeset is supported on HP-UX. The HP-UX proprietary greek8 codeset has no registered CCSID and is not supported.

i5/OS

Code page:

875 Does not convert to code page 5349

Windows

Code page:

1253 Does not convert to code page 737

5349 Does not convert to code page 737

All non-client platforms support conversion between their native CCSIDs, the native CCSIDs of the other platforms with the following exceptions.

Turkish

Details of CCSIDs and CCSID conversion for Turkish

The following table shows the native CCSIDs for Turkish on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	1026
HP-UX	920 (see note)
Windows	857, 1254, 5350
AIX, OVMS, NSS, Solaris, Linux, Tru64	920
Apple client	1281

Note: Only the ISO codeset is supported on HP-UX. The HP-UX proprietary turkish8 codeset has no registered CCSID and is not supported.

i5/OS

Code page:

1026 Does not convert to code page 5350

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

Hebrew

The following table shows the native CCSIDs for Hebrew on supported platforms:

Platform	Native CCSIDs
z/OS	424, 803, 4899, 12712
i5/OS	424
AIX	916, 9048
HP-UX	916 (see note)
Windows	1255, 5351
OVMS, NSS, Solaris, Linux, Tru64	916

Note: Only the ISO codeset is supported on HP-UX. The HP-UX proprietary greek8 codeset has no registered CCSID and is not supported.

z/OS

Code page:

- 424 Does not convert to code pages 867, 4899, 9048, 12712
- 803 Does not convert to code pages 867, 4899, 5351, 9048, 12712
- 4899 Does not convert to code pages 424, 803, 856, 862, 916, 1255
- 12712 Does not convert to code pages 424, 803, 856, 916, 1255

i5/OS

Code page:

- 424 Does not convert to code pages 803, 867, 4899, 5351, 9048, 12712
Code page 424 also converts to and from CCSID 4952, which is a variant of 856.

AIX

Code page:

- 916 Does not convert to code pages 867, 4899, 9048, 12712
- 9048 Does not convert to code pages 424, 803, 856, 862, 916, 1255

Windows

Code page:

- 1255 Does not convert to code pages 867, 4899, 9048, 12712
- 5351 Does not convert to code page 803

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

Arabic

Details of CCSIDs and CCSID conversion for Arabic

The following table shows the native CCSIDs for Arabic on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	420
AIX	1046, 1089
HP-UX	1089 (see note)
Windows	720, 864, 1256, 5352
OVMS, NSS, Solaris, Linux, Tru64	1089
Note: Only the ISO codeset is supported on HP-UX. The HP-UX proprietary arabic8 codeset has no registered CCSID and is not supported.	

i5/OS

Code page:

- 420 Does not convert to code page 5352

OS/2

Code page:

864 Does not convert to code page 720

HP-UX, Solaris, Linux, OVMS, NSS, Tru64

Code page:

1089 Does not convert to code page 720

Windows

Code page:

720 Does not convert to code pages 1089, 5352

5352 Does not convert to code page 720

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

Farsi

Details of CCSIDs and CCSID conversion for Farsi

The following table shows the native CCSIDs for Farsi on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	1097
AIX, OVMS, HP-UX, NSS, Solaris, Linux, Tru64, Windows	1098 (see note)
Note: The native CCSID for these platforms has not been standardized and may change.	

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms.

Urdu

Details of CCSIDs and CCSID conversion for Urdu

The following table shows the native CCSIDs for Urdu on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	918
Windows	868
AIX, HP-UX, OVMS, NSS, Solaris, Linux, Tru64	1006

i5/OS

Code page:

918 Does not convert to code page 1006

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

Thai

Details of CCSIDs and CCSID conversion for Thai

The following table shows the native CCSIDs for Thai on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	838
AIX, OVMS, HP-UX, NSS, Solaris, Linux, Tru64, Windows	874 (see note)
Note: The native CCSID for these platforms has not been standardized and may change.	

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms.

Lao

Details of CCSIDs and CCSID conversion for Lao

The following table shows the native CCSIDs for Lao on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	1132
AIX, OVMS, HP-UX, NSS, Solaris, Linux, Tru64, Windows	1133

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms.

Vietnamese

Details of CCSIDs and CCSID conversion for Vietnamese

The following table shows the native CCSIDs for Vietnamese on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	1130
Windows	1258, 5354
AIX, OVMS, HP-UX, NSS, Solaris, Linux, Tru64	1129

i5/OS

Code page:

1130 Does not convert to code pages 1129, 5354

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

Japanese Latin SBCS

Details of CCSIDs and CCSID conversion for Japanese Latin SBCS

The following table shows the native CCSIDs for Japanese Latin SBCS on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	1027
AIX	932, 5050, 33722 (see Note 1)
Windows	932, 943 (see Notes 2 and 3)
OVMS, Linux, NSS, Solaris	943, 5050
HP-UX	Not known
Tru64	943, 954, 5050, 33722

Note:

1. 5050 and 33722 are CCSIDs related to base code page 954 on AIX. The CCSID reported by the operating system is 33722.
2. Windows NT uses code page 932 but this is best represented by the CCSID of 943. However, not all platforms of WebSphere MQ support this CCSID.
On WebSphere MQ for Windows CCSID 932 is used to represent code page 932, but a change to file `../conv/table/ccsid.tbl` can be made which changes the CCSID used to 943.
3. WebSphere MQ does not support code pages based on the JIS X 0213 (JIS2004) standard.

z/OS

Code page:

1027 Does not convert to code pages 932, 942, 943, 954, 5050, 33722

i5/OS

Code page:

1027 Does not convert to code page 932

AIX

Code page:

932 Does not convert to code page 1027

5050 Does not convert to code page 1027

33722 Does not convert to code page 1027

Linux

Code page:

943 Does not convert to code page 1027

5050 Does not convert to code page 1027

Solaris

Code page:

943 Does not convert to code page 1027

5050 Does not convert to code page 1027

NSS

Code page:

943 Does not convert to code page 1027

5050 Does not convert to code page 1027

Tru64

Code page:

943 Does not convert to code page 1027

954 Does not convert to code page 1027

5050 Does not convert to code page 1027

33722 Does not convert to code page 1027

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

Japanese Katakana SBCS

Details of CCSIDs and CCSID conversion for Japanese Katakana SBCS

The following table shows the native CCSIDs for Japanese Katakana SBCS on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	290
HP-UX	897
AIX	932, 5050, 33722 (see Note 1)
Windows	932, 943 (see Notes 2 and 3)
OVMS, Linux, NSS, Solaris	943, 5050
Tru64	943, 954, 5050, 33722

Note:

- 5050 and 33722 are CCSIDs related to base code page 954 on AIX. The CCSID reported by the operating system is 33722.
- Windows NT uses code page 932 but this is best represented by the CCSID of 943. However, not all platforms of WebSphere MQ support this CCSID.
On WebSphere MQ for Windows CCSID 932 is used to represent code page 932, but a change to file `../conv/table/ccsid.tbl` can be made which changes the CCSID used to 943.
- WebSphere MQ does not support code pages based on the JIS X 0213 (JIS2004) standard.
- In addition to the above conversions, the WebSphere MQ products on AIX, HP-UX, Solaris, Linux and Tru64 support conversion from CCSID 897 to CCSIDs 37, 273, 277, 278, 280, 284, 285, 290, 297, 437, 500, 819, 850, 1027, and 1252.

z/OS

Code page:

290 Does not convert to code pages 932, 943, 954, 5050, 33722

i5/OS

Code page:

290 Does not convert to code page 932

AIX

Code page:

932 Does not convert to code pages 290, 897

5050 Does not convert to code pages 290, 897

33722 Does not convert to code pages 290, 897

HP-UX

Code page:

897 Does not convert to code pages 932, 943, 954, 5050, 33722

Linux

Code page:

943 Does not convert to code pages 290, 897

5050 Does not convert to code pages 290, 897

Solaris

Code page:

943 Does not convert to code pages 290, 897

5050 Does not convert to code pages 290, 897

DEC-OVMS

Code page:

943 Does not convert to code pages 290, 897, 932, 954, 5050, 33722

954 Does not convert to code pages 290, 897, 943

OVMS, NSS

Code page:

943 Does not convert to code pages 290, 897

5050 Does not convert to code pages 290, 897

Tru64

Code page:

- 943 Does not convert to code pages 290, 897
- 954 Does not convert to code pages 290, 897
- 5050 Does not convert to code pages 290, 897
- 33722 Does not convert to code pages 290, 897

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

Japanese Kanji/ Latin Mixed

Details of CCSIDs and CCSID conversion for Japanese Kanji/Latin Mixed.

The following table shows the native CCSIDs for Japanese Kanji/ Latin Mixed on supported platforms:

Platform	Native CCSIDs
i5/OS, z/OS	1399, 5035 (see Note 1)
AIX	932, 5050, 33722 (see Note 2)
HP-UX	932, 954, 5039 (see Note 3)
Windows	932, 943 (see Notes 4 and 5)
OVMS, Linux, NSS, Solaris	943, 5050
Tru64	943, 954, 5050, 33722

Note:

1. 5035 is a CCSID related to code page 939
2. 5050 and 33722 are CCSIDs related to base code page 954 on AIX. The CCSID reported by the operating system is 33722.
3. Code sets japan15 and SJIS on HP-UX are represented by CCSID 932. These have a few DBCS characters having different representations in SJIS so 932 may be converted incorrectly if the conversion is not performed on an HP-UX system. WebSphere MQ for HP-UX supports 5039, the correct CCSID for HP SJIS. A change to file `/var/mqm/conv/ccsid.tbl` can be made to change the CCSID used from 932 to 5039.
4. Windows NT uses code page 932 but this is best represented by the CCSID of 943. However, not all platforms of WebSphere MQ support this CCSID.
On WebSphere MQ for Windows CCSID 932 is used to represent code page 932, but a change to file `../conv/table/ccsid.tbl` can be made which changes the CCSID used to 943.
5. WebSphere MQ does not support code pages based on the JIS X 0213 (JIS2004) standard.

z/OS

Code page:

- 1399 Does not convert to code pages 954, 5035, 5050, 33722
- 5035 Does not convert to code pages 954, 1399, 5050, 33722

i5/OS

Code page:

1399 Does not convert to code page 5039

5035 Does not convert to code page 5039

HP-UX

Code page:

932 Does not convert to code pages 942, 943, 1399

954 Does not convert to code pages 942, 943, 1399

5039 Does not convert to code pages 942, 943, 1399

OVMS, NSS

Code page:

943 Does not convert to code page 1399

5050 Does not convert to code page 1399

Tru64

Code page:

943 Does not convert to code page 1399

954 Does not convert to code page 1399

5050 Does not convert to code page 1399

33722 Does not convert to code page 1399

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

Japanese Kanji/ Katakana Mixed

Details of CCSIDs and CCSID conversion for Japanese Kanji/Katakana Mixed.

The following table shows the native CCSIDs for Japanese Kanji/ Katakana Mixed on supported platforms:

Platform	Native CCSIDs
z/OS	1390, 5026 (see Note 1)
i5/OS	5026 (see Note 1)
AIX	932, 5050, 33722 (see Note 2)
HP-UX	932, 954, 5039 (see Note 3)
Windows	932, 943 (see Notes 4 and 5)
OVMS, Linux, NSS, Solaris	943, 5050
Tru64	943, 954, 5050, 33722

Platform	Native CCSIDs
<p>Note:</p> <ol style="list-style-type: none"> 1. CCSID 1390 does not accept lower case characters. 5026 is a CCSID related to code page 930. CCSID 5026 is the CCSID reported on i5/OS when the Japanese Katakana (DBCS) feature is selected. 2. 5050 and 33722 are CCSIDs related to base code page 954 on AIX. The CCSID reported by the operating system is 33722. 3. Code sets japan15 and SJIS on HP-UX are represented by CCSID 932. These have a few DBCS characters having different representations in SJIS so 932 may be converted incorrectly if the conversion is not performed on an HP-UX system. WebSphere MQ for HP-UX supports 5039, the correct CCSID for HP SJIS. A change to file <code>/var/mqm/conv/ccsid.tbl</code> can be made to change the CCSID used from 932 to 5039. 4. Windows NT uses code page 932 but this is best represented by the CCSID of 943. However, not all platforms of WebSphere MQ support this CCSID. On WebSphere MQ for Windows, CCSID 932 is used to represent code page 932, but a change to file <code>./conv/table/ccsid.tbl</code> can be made that changes the CCSID used to 943. 5. WebSphere MQ does not support code pages based on the JIS X 0213 (JIS2004) standard. 	

z/OS

Code page:

1390 Does not convert to code pages 954, 5026, 5050, 33722

Does not accept lower case characters.

5026 Does not convert to code pages 954, 1390, 5050, 33722

i5/OS

Code page:

5026 Does not convert to code pages 1390, 5039

HP-UX

Code page:

932 Does not convert to code pages 942, 943, 1390

954 Does not convert to code pages 942, 943, 1390

5039 Does not convert to code pages 942, 943, 1390

OVMS, NSS

Code page:

943 Does not convert to code page 1390

5050 Does not convert to code page 1390

Tru64

Code page:

943 Does not convert to code page 1390

- 954 Does not convert to code page 1390
- 5050 Does not convert to code page 1390
- 33722 Does not convert to code page 1390

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

Korean

Details of CCSIDs and CCSID conversion for Korean.

The following table shows the native CCSIDs for Korean on supported platforms:

Platform	Native CCSIDs
z/OS, i5/OS	933, 1364
AIX, OVMS, HP-UX, Linux, NSS, Solaris	970
Windows	949, 1363
Tru64	970, 1363

z/OS

Code page:

- 933 Does not convert to code page 970
- 1364 Does not convert to code page 970

HP-UX

Code page:

- 970 Does not convert to code pages 949, 1363, 1364

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

Simplified Chinese

Details of CCSIDs and CCSID conversion for Simplified Chinese.

The following table shows the native CCSIDs for Simplified Chinese on supported platforms:

Platform	Native CCSIDs
z/OS	935, 1388
i5/OS	935, 1388
AIX	1383, 1386
HP-UX	1381 (see Note 1)
Windows	1381, 1386(see Note 2)
OVMS, Linux, NSS, Solaris, Tru64	1383

Platform	Native CCSIDs
<p>Note:</p> <ol style="list-style-type: none"> Code sets prc15 and hp15CN on HP-UX are represented by CCSID 1381. Windows uses code page 936 but this is best represented by the CCSID of 1386. However, not all platforms of WebSphere MQ support this CCSID. On WebSphere MQ for Windows CCSID 1381 is used to represent code page 936, but a change to file <code>../conv/table/ccsid.tbl</code> can be made which changes the CCSID used to 1386. WebSphere MQ supports phase one of the Chinese GB18030 standard. On z/OS, Linux, Windows, and Solaris, conversion support is provided between Unicode (UTF-8 and UCS-2) and CCSID 1388 (EBCDIC with GB18030 extensions), Unicode (UTF-8 and UCS-2) and CCSID 5488 (GB18030 phase one), and between CCSID 1388 and CCSID 5488. <p>Note:</p> <p>On i5/OS, support is provided by the operating system for conversion between Unicode (UTF-8 and UCS-2) and CCSID 1388 (EBCDIC with GB18030 extensions).</p> <p>On HP-UX there is currently no support available on the HP11 operating system for GB18030. On HP11i, patch PHCO_26456 provides conversion support between GB18030 (CCSID 5488) and Unicode. Support is not provided for the conversion between GB18030 and 1388 (EBCDIC).</p>	

z/OS

Code page:

935 Does not convert to code page 1383

1388 Does not convert to code page 1383

HP-UX

Code page:

1381 Does not convert to code pages 1383, 1386, 1388

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

Traditional Chinese

Details of CCSIDs and CCSID conversion for Traditional Chinese.

The following table shows the native CCSIDs for Traditional Chinese on supported platforms:

Platform	Native CCSIDs
z/OS, i5/OS	937
HP-UX	938, 950, 964 (see Note)
Tru64, Windows	950
AIX, OVMS, NSS, Solaris, Linux	950, 964
Note: Code set roc15 on HP-UX is represented by CCSID 938.	

z/OS

Code page:

937 Does not convert to code page 964

1388 Does not convert to code page 1383

HP-UX

Code page:

938 Does not convert to code page 948

950 Does not convert to code page 948

964 Does not convert to code page 948

OVMS, Linux, Solaris

Code page:

964 Does not convert to code page 938

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS conversion support

Table 107. WebSphere MQ for z/OS CCSID conversion support

CCSID	Converts to and from CCSIDs
37	256, 273, 275, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 720, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-866, 869-871, 874-875, 880, 897, 903-905, 912, 914-916, 920-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1097, 1100, 1112, 1114-1115, 1122, 1124, 1126, 1130-1132, 1137, 1140-1149, 1200, 1208, 1250-1255, 1257-1258, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5210-5211, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 16804, 17248, 17584, 25473, 25479, 25480, 25617, 25619, 25664, 28709
256	37, 273, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 737, 775, 819, 833, 836, 838, 850, 852, 857, 860-866, 869-871, 875, 880, 905, 1025-1027, 1112, 1122, 1200, 1208, 1251-1252, 1275, 4386, 4929, 4932, 4934, 4946, 4948, 4953, 4960, 4971, 5123, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 13121, 13488, 16804, 17248, 17584, 28709
259	437, 808, 850-852, 855-858, 860-865, 867, 869, 872, 874, 899, 901-902, 915, 1098, 1161-1162, 1200, 1208, 1250-1258, 4946, 4948, 4951-4953, 4960, 4970, 5346, 5348, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584
273	37, 256, 277-278, 280, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1250, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951-4953, 4960, 4970-4971, 5012, 5123, 5346, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
274	500, 1047
275	37, 437, 500, 819, 850, 1047, 1200, 1208, 1252, 4946, 5348, 8229, 13488, 17584, 28709

Table 107. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
277	37, 256, 273, 278, 280, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
278	37, 256, 273, 277, 280, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
280	37, 256, 273, 277-278, 280, 285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
281	1047
282	500, 1047, 1200, 1208, 13488, 17584
284	37, 256, 273, 277-278, 280, 285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
285	37, 256, 273, 277-278, 280, 284, 290, 297, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
290	37, 256, 273, 277-278, 280, 284-285, 297, 367, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 860-865, 870-871, 895-897, 1009, 1025-1027, 1040-1043, 1088, 1112, 1122, 1139, 1200, 1208, 1252, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4960, 4992, 5123, 8229, 8482, 9025, 9044, 9049, 9056, 13121, 13488, 17248, 17584, 25473, 25617, 25619, 25664, 28709
293	1200, 1208, 13488, 17584
297	37, 256, 273, 277-278, 280, 284-285, 290, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1100, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
300	301, 941, 1200, 1208, 1351, 4396, 8492, 13488, 16684, 17584
301	300, 941, 1200, 1208, 1351, 4396, 8492, 13488, 16684, 17584
367	37, 256, 273, 277-278, 280, 284, 290, 297, 500, 819, 833, 836, 850, 871, 875, 1009, 1026-1027, 1041, 1088, 1115, 1126, 1200, 1208, 4386, 4929, 4932, 4946, 4971, 5123, 5211, 8229, 8482, 9025, 13121, 13488, 17584, 25617, 25664, 28709
420	37, 256, 424, 437, 500, 720, 737, 775, 819, 850, 852, 857, 860-865, 1008, 1046, 1089, 1098, 1112, 1122, 1127, 1200, 1208, 1252, 1256, 4946, 4948, 4953, 4960, 5104, 5142, 5352, 8229, 8612, 9044, 9049, 9056, 9238, 13488, 16804, 17248, 17584, 28709

Table 107. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
423	37, 256, 273, 277-278, 280, 284-285, 297, 437, 500, 737, 775, 813, 819, 838, 850-852, 857, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1009, 1025-1027, 1041-1043, 1112, 1122, 1200, 1208, 1252-1253, 1280, 4909, 4934, 4946, 4948, 4953, 4960, 4970-4971, 5012, 5123, 8229, 9030, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 28709
424	37, 256, 420, 437, 500, 737, 775, 803, 819, 836, 850, 852, 856-857, 860-865, 916, 1112, 1122, 1200, 1208, 1252, 1255, 4932, 4946, 4948, 4952-4953, 4960, 5012, 5351, 8229, 8612, 9044, 9049, 9056, 13488, 16804, 17248, 17584, 28709
437	37, 256, 259, 273, 275, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-863, 865-866, 869-871, 874-875, 880, 897, 903, 905, 912, 914-916, 920-924, 1025-1027, 1040-1043, 1047, 1051, 1097, 1098, 1114-1115, 1126, 1140-1149, 1200, 1208, 1252, 1257, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5210-5211, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 28709
500	37, 256, 273-275, 277-278, 280, 282, 284-285, 290, 297, 367, 420, 423-424, 437, 737, 775, 813, 819, 833, 836, 838, 850-852, 855-858, 860-866, 869-871, 874-875, 880, 891, 895, 897, 903-905, 912, 914-916, 920-924, 1004, 1009-1021, 1023, 1025-1027, 1040-1043, 1046-1047, 1051, 1088-1089, 1097, 1100-1107, 1112, 1114-1115, 1122, 1124-1126, 1129-1133, 1137, 1140-1149, 1200, 1208, 1250-1258, 1275, 1280-1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951-4953, 4960, 4970-4971, 5012, 5123, 5142, 5210-5211, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 9238, 13121, 13488, 16804, 17248, 17584, 25473, 25479, 25480, 25617, 25619, 25664, 28709
720	37, 420, 864, 1200, 1208, 1256, 4960, 8229, 8612, 9056, 13488, 16804, 17248, 17584, 28709
737	37, 256, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 833, 836, 838, 850, 869-871, 875, 880, 905, 1025-1027, 1097, 1200, 1208, 1252-1253, 1280, 4386, 4909, 4929, 4932, 4934, 4946, 4971, 5123, 8229, 8482, 8612, 9025, 9030, 9061, 13121, 13488, 16804, 17584, 28709
775	37, 256, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 833, 836, 838, 850, 870-871, 875, 880, 905, 1025-1027, 1097, 1112, 1122, 1200, 1208, 1252, 1257, 4386, 4929, 4932, 4934, 4946, 4971, 5123, 8229, 8482, 8612, 9025, 9030, 13121, 13488, 16804, 17584, 28709
803	424, 819, 850, 856, 862, 916, 1200, 1208, 1252, 1255, 4946, 4952, 5012, 13488, 17584
806	1200, 1208, 13488, 17584
808	259, 858-859, 872, 923-924, 1140, 1148, 1153-1154, 1200, 1208, 5347, 5348, 13488, 17584
813	37, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252-1253, 1280, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5349, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
819	37, 256, 273, 275, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 803, 813, 833, 836, 838, 850, 852, 855, 857-858, 860-861, 863-866, 869-871, 874-875, 880, 897, 903, 905, 912, 914-916, 920-924, 1004, 1025-1027, 1041-1043, 1047, 1051, 1088-1089, 1097, 1098, 1112, 1114, 1122-1123, 1126, 1130, 1132, 1137, 1140-1149, 1200, 1208, 1250-1255, 1257-1258, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5210, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 16804, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709

Table 107. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
833	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 437, 500, 737, 775, 819, 836, 850, 852, 855, 857, 860-865, 870-871, 891, 1009, 1025-1027, 1040-1043, 1088, 1112, 1122, 1126, 1200, 1208, 1252, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4960, 5123, 8229, 8482, 9025, 9044, 9049, 9056, 13121, 13488, 17248, 17584, 25617, 25619, 25664, 28709
834	926, 951, 1200, 1208, 1362, 4930, 9026, 13488, 17584
835	927, 947, 1200, 1208, 4931, 9027, 13488, 17584, 21427
836	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 424, 437, 500, 737, 775, 819, 833, 850, 852, 855, 857, 870-871, 875, 903, 1009, 1025-1027, 1040-1043, 1088, 1112, 1114-1115, 1122, 1200, 1208, 1252, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4971, 5123, 5210-5211, 8229, 8482, 9025, 9044, 9049, 13121, 13488, 17584, 25479, 25617, 25619, 25664, 28709
837	928, 1200, 1208, 1380, 1385, 4933, 13488, 17584
838	37, 256, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 775, 813, 819, 850, 852, 857, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1112, 1122, 1200, 1208, 1252, 4909, 4934, 4946, 4948, 4953, 4960, 4970-4971, 5012, 5123, 8229, 9030, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 28709
848	924, 1148, 1158, 1200, 1208, 5347, 13488, 17584
849	924, 1148, 1154, 1200, 1208, 5347, 13488, 17584
850	37, 256, 259, 273, 275, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 737, 775, 803, 813, 819, 833, 836, 838, 852, 855-858, 860-866, 869-871, 874-875, 880, 897, 903, 905, 912, 914-916, 920-924, 1004, 1025-1027, 1040-1043, 1047, 1051, 1088-1089, 1097, 1098, 1100, 1112, 1114, 1122, 1126, 1130, 1132, 1140-1149, 1200, 1208, 1250-1257, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951-4953, 4960, 4970-4971, 5012, 5123, 5210, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 16804, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
851	259, 423, 500, 875, 1200, 1208, 4971, 13488, 17584
852	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 836, 838, 850, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1040-1043, 1047, 1088, 1097, 1200, 1208, 1250, 1252, 1282, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5346, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 25664, 28709
855	37, 259, 273, 277-278, 280, 284-285, 290, 297, 437, 500, 819, 833, 836, 850, 852, 857, 866, 870-871, 878, 880, 912, 915, 1025-1027, 1040-1043, 1088, 1200, 1208, 1250-1252, 1283, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 5123, 5346, 5347, 8229, 8482, 9025, 9044, 9049, 13121, 13488, 17584, 25617, 25619, 25664, 28709
856	259, 273, 424, 500, 803, 850, 862, 916, 1200, 1208, 1255, 4946, 4952, 5012, 5351, 13488, 17584
857	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1040-1043, 1088, 1097, 1200, 1208, 1252, 1254, 1281, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5350, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 25664, 28709
858	37, 259, 273, 277-278, 280, 284-285, 297, 437, 500, 808, 819, 850, 860-861, 865, 871-872, 901-902, 923-924, 1047, 1051, 1140-1149, 1153-1157, 1160-1162, 1164, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
859	808, 872, 901-902, 1153-1157, 1160-1162, 1164, 1200, 1208, 13488, 17584

Table 107. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
860	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 838, 850, 852, 857-858, 861, 863, 865, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 923-924, 1025-1027, 1041-1043, 1097, 1140, 1145-1146, 1148, 1200, 1208, 1252, 4386, 4909, 4929, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 28709
861	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 838, 850, 852, 857-858, 860, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 923-924, 1025-1027, 1041-1043, 1097, 1148, 1149, 1200, 1208, 1252, 4386, 4909, 4929, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 28709
862	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 803, 833, 838, 850, 856, 870-871, 875, 880, 905, 916, 1025-1027, 1097, 1200, 1208, 1252, 1255, 4386, 4929, 4934, 4946, 4952, 4971, 5012, 5123, 5351, 8229, 8482, 8612, 9025, 9030, 12712, 13121, 13488, 16804, 17584, 28709
863	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 838, 850, 852, 857, 860-861, 865, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1041-1043, 1051, 1097, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 28709
864	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 720, 819, 833, 838, 850, 870-871, 875, 880, 905, 918, 1008, 1025-1027, 1046, 1089, 1097, 1127, 1200, 1208, 1252, 1256, 4386, 4929, 4934, 4946, 4960, 4971, 5104, 5123, 5142, 5352, 8229, 8482, 8612, 9025, 9030, 9056, 9238, 13121, 13488, 16804, 17248, 17584, 28709
865	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 819, 833, 838, 850, 858, 860, 863, 870-871, 875, 880, 905, 923-924, 1025-1027, 1097, 1142-1143, 1148, 1200, 1208, 1252, 4386, 4929, 4934, 4946, 4971, 5123, 5348, 8229, 8482, 8612, 9025, 9030, 13121, 13488, 16804, 17584, 28709
866	37, 256, 437, 500, 819, 850, 855, 870, 878, 880, 915, 1025, 1200, 1208, 1251-1252, 1283, 4946, 4951, 5347, 8229, 13488, 17584, 28709
867	259, 1153-1155, 1160, 1200, 1208, 4899, 5351, 9048, 12712, 13488, 17584
868	918, 1006, 1200, 1208, 13488, 17584
869	37, 256, 259, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 813, 819, 838, 850, 852, 857, 860-861, 863, 870-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252-1254, 1280, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5349, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
870	37, 256, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-866, 869, 871, 874-875, 880, 897, 903, 912, 915-916, 920, 1009, 1025-1027, 1040-1043, 1047, 1088, 1112, 1122, 1200, 1208, 1250, 1252, 1282, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5346, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
871	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-865, 869, 870, 874-875, 880, 897, 903, 912, 916, 920, 923-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1112, 1122, 1140-1149, 1200, 1208, 1252, 1275, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5348, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709

Table 107. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
872	259, 808, 858-859, 923-924, 1140-1149, 1153-1155, 1200, 1208, 5347, 5348, 13488, 17584
874	37, 259, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
875	37, 256, 273, 277-278, 280, 284-285, 297, 367, 423, 437, 500, 737, 775, 813, 819, 836, 838, 850-852, 857, 860-865, 869-871, 874, 880, 897, 903, 912, 916, 920, 1009, 1025-1027, 1041-1043, 1047, 1088, 1112, 1122, 1200, 1208, 1252-1253, 1280, 4909, 4932, 4934, 4946, 4948, 4953, 4960, 4970-4971, 5012, 5123, 5349, 8229, 9030, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
878	855, 866, 880, 915, 1025, 1131, 1200, 1208, 1251, 1283, 4951, 5347, 13488, 17584
880	37, 256, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 775, 813, 819, 838, 850, 852, 855, 857, 860-866, 869-871, 874-875, 878, 897, 903, 912, 915-916, 920, 1009, 1025-1027, 1041-1043, 1112, 1122, 1200, 1208, 1251-1252, 1283, 4909, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5347, 8229, 9030, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 28709
891	500, 833, 1088, 1200, 1208, 4929, 9025, 13121, 13488, 17584, 25664
895	290, 500, 1027, 1041, 1200, 1208, 4386, 5123, 8482, 13488, 17584, 25617
896	290, 1027, 1041, 1200, 1208, 4386, 4992, 5123, 8482, 13488, 17584, 25617
897	37, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252, 4386, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 8229, 8482, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
899	259
901	259, 858-859, 902, 923-924, 1140, 1148, 1156-1157, 1200, 1208, 5348, 5353, 13488, 17584
902	259, 858-859, 901, 923-924, 1140, 1148, 1156-1157, 1200, 1208, 5348, 5353, 13488, 17584
903	37, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 836, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 912, 916, 920, 1025-1027, 1041-1043, 1115, 1200, 1208, 1252, 4909, 4932, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5211, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
904	37, 500, 1114, 1200, 1208, 5210, 8229, 13488, 17584, 25480, 28709
905	37, 256, 437, 500, 737, 775, 819, 850, 852, 857, 860-865, 920, 1026, 1112, 1122, 1200, 1208, 1252, 1254, 1281, 4946, 4948, 4953, 4960, 8229, 9044, 9049, 9056, 13488, 17248, 17584, 28709
912	37, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 916, 920, 1025-1027, 1041-1043, 1047, 1200, 1208, 1250, 1252, 1282, 4909, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5346, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
914	37, 437, 500, 819, 850, 1200, 1208, 1252, 1257, 4946, 8229, 13488, 17584, 28709
915	37, 259, 437, 500, 819, 850, 855, 866, 870, 878, 880, 1025, 1131, 1200, 1208, 1251-1252, 1283, 4946, 4951, 5347, 8229, 13488, 17584, 28709

Table 107. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
916	37, 273, 277-278, 280, 284-285, 297, 423-424, 437, 500, 803, 813, 819, 838, 850, 852, 856-857, 860-863, 869-871, 874-875, 880, 897, 903, 912, 920, 1025-1027, 1041-1043, 1200, 1208, 1252, 1255, 4909, 4934, 4946, 4948, 4952-4953, 4970-4971, 5012, 5123, 5351, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
918	864, 868, 1006, 1200, 1208, 4960, 9056, 13488, 17248, 17584
920	37, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 1025-1026, 1200, 1208, 1252, 1254, 1281, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5350, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 28709
921	37, 437, 500, 819, 850, 922, 1112, 1122, 1200, 1208, 1252, 1257, 4946, 5353, 8229, 13488, 17584, 28709
922	37, 437, 500, 819, 850, 921, 1112, 1122, 1200, 1208, 1252, 1257, 4946, 5353, 8229, 13488, 17584, 28709
923	37, 273, 277-278, 280, 284-285, 297, 437, 500, 808, 819, 850, 858, 860-861, 865, 871-872, 901-902, 924, 1047, 1051, 1140-1149, 1153-1158, 1160-1162, 1164, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
924	37, 273, 277-278, 280, 284-285, 297, 437, 500, 808, 819, 848-850, 858, 860-861, 865, 871-872, 901-902, 923, 1047, 1051, 1140-1149, 1153-1157, 1160-1164, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
926	834, 951, 9026
927	835, 947, 1200, 1208, 4931, 9027, 13488, 17584, 21427
928	837, 1200, 1208, 1380, 13488, 17584
930	931-932, 939, 942-943, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
931	930, 932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
932	930-931, 939, 942-943, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
933	934, 944, 949, 1200, 1208, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13488, 13651, 17317, 17584, 25510, 25520, 25525, 29616, 29621, 33717, 37813
934	933, 949, 5029, 5045, 5460, 9125, 13221, 17317, 25510, 25525, 29621, 33717, 37813
935	936, 946, 1200, 1208, 1381, 1386, 1388, 5031, 5477, 5482, 5484, 9127, 13223, 13488, 17584, 25512
936	935, 946, 1381, 5031, 5477, 5484, 9127, 13223, 25512
937	938, 948, 950, 1200, 1208, 1370, 5033, 5046, 9142, 13488, 17584, 25514, 25524, 29620
938	937, 950, 1370, 5033, 5046, 9142, 25514
939	930-932, 942-943, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
941	300-301, 1200, 1208, 1351, 4396, 8492, 13488, 16684, 17584
942	930-932, 939, 943, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796

Table 107. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
943	930-932, 939, 942, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
944	933, 949, 1200, 1208, 5029, 5045, 5460, 9125, 13221, 13488, 17317, 17584, 25520, 25525, 29616, 29621, 33717, 37813
946	935-936, 1200, 1208, 5031, 5484, 9127, 13223, 13488, 17584, 25512
947	835, 927, 1200, 1208, 4931, 9027, 13488, 17584, 21427
948	937, 950, 1200, 1208, 1370, 5033, 5046, 9142, 13488, 17584, 25524, 29620
949	933-934, 944, 1200, 1208, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13488, 13651, 17317, 17584, 25510, 25520, 25525, 29616, 29621, 33717, 37813
950	937-938, 948, 1200, 1208, 1370, 5033, 5046, 9142, 13488, 17584, 25514, 25524, 29620
951	834, 926, 1200, 1208, 1362, 4930, 9026, 13488, 17584
1004	500, 819, 850, 1200, 1208, 4946, 13488, 17584
1006	868, 918, 1200, 1208, 13488, 17584
1008	420, 864, 1200, 1208, 4960, 5104, 8612, 9056, 13488, 16804, 17248, 17584
1009	37, 273, 277-278, 280, 284, 290, 297, 367, 423, 500, 833, 836, 870-871, 875, 880, 1025-1026, 1200, 1208, 4386, 4929, 4932, 4971, 8229, 8482, 9025, 13121, 13488, 17584, 28709
1010	500, 1200, 1208, 13488, 17584
1011	500, 1200, 1208, 13488, 17584
1012	500, 1200, 1208, 13488, 17584
1013	500, 1140, 1200, 1208, 13488, 17584
1014	500, 1200, 1208, 13488, 17584
1015	500, 1200, 1208, 13488, 17584
1016	500, 1200, 1208, 13488, 17584
1017	500, 1200, 1208, 13488, 17584
1018	500, 1200, 1208, 13488, 17584
1019	500, 1200, 1208, 13488, 17584
1020	500
1021	500
1023	500
1025	37, 256, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-866, 869-871, 874-875, 878, 880, 897, 903, 912, 915-916, 920, 1009, 1026-1027, 1040-1043, 1051, 1088, 1112, 1122, 1131, 1200, 1208, 1251-1252, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5347, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
1026	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-865, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1009, 1025, 1027, 1040-1043, 1047, 1088, 1112, 1122, 1200, 1208, 1252, 1254, 1281, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5350, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709

Table 107. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
1027	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-865, 869-871, 874-875, 880, 895-897, 903, 912, 916, 1025-1026, 1040-1043, 1047, 1088, 1112, 1122, 1139, 1200, 1208, 1252, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 4992, 5012, 5123, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
1040	37, 273, 277-278, 280, 284-285, 290, 297, 437, 500, 833, 836, 850, 852, 855, 857, 870-871, 1025-1027, 1041-1043, 1088, 1200, 1208, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 5123, 8229, 8482, 9025, 9044, 9049, 13121, 13488, 17584, 25617, 25619, 25664, 28709
1041	37, 273, 277-278, 280, 284-285, 290, 297, 367, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 895-897, 903, 912, 916, 1025-1027, 1040, 1042-1043, 1088, 1200, 1208, 1252, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 4992, 5012, 5123, 8229, 8482, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 17584, 25473, 25479, 25617, 25619, 25664, 28709
1042	37, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 1025-1027, 1040, 1041, 1043, 1088, 1200, 1208, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 8229, 8482, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 17584, 25473, 25479, 25617, 25619, 25664, 28709
1043	37, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 1025-1027, 1040, 1041, 1042, 1088, 1114, 1200, 1208, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5210, 8229, 8482, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 17584, 25473, 25479, 25617, 25619, 25664, 28709
1046	420, 500, 864, 1089, 1127, 1200, 1208, 1256, 4960, 5142, 5352, 8612, 9056, 9238, 13488, 16804, 17248, 17584
1047	37, 273-275, 277-278, 280, 281, 282, 284-285, 297, 437, 500, 819, 850, 852, 858, 870-871, 875, 912, 923-924, 1026-1027, 1140-1149, 1200, 1208, 1252, 1254, 4946, 4948, 4971, 5123, 8229, 9044, 13488, 17584, 28709
1051	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 871, 923-924, 1025, 1097, 1140-1149, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1088	37, 273, 277-278, 280, 284-285, 290, 297, 367, 500, 819, 833, 836, 850, 852, 855, 857, 870-871, 875, 891, 1025-1027, 1040-1043, 1126, 1200, 1208, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4971, 5123, 8229, 8482, 9025, 9044, 9049, 13121, 13488, 17584, 25617, 25619, 25664, 28709
1089	420, 500, 819, 850, 864, 1046, 1127, 1200, 1208, 1256, 4946, 4960, 5142, 5352, 8612, 9056, 9238, 13488, 16804, 17248, 17584
1097	37, 437, 500, 737, 775, 819, 850, 852, 857, 860-865, 1051, 1098, 1112, 1122, 1200, 1208, 1252, 4946, 4948, 4953, 4960, 8229, 9044, 9049, 9056, 13488, 17248, 17584, 28709
1098	259, 420, 437, 819, 850, 1097, 1200, 1208, 1252, 4946, 8612, 13488, 16804, 17584
1100	37, 273, 277-278, 280, 284-285, 297, 500, 850, 4946, 8229, 28709
1101	500
1102	500
1103	500
1104	500
1105	500
1106	500

Table 107. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
1107	500
1112	37, 256, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 775, 819, 833, 836, 838, 850, 870-871, 875, 880, 905, 921-922, 1025-1027, 1097, 1122, 1200, 1208, 1252, 1257, 4386, 4929, 4932, 4934, 4946, 4971, 5123, 5353, 8229, 8482, 8612, 9025, 9030, 13121, 13488, 16804, 17584, 28709
1114	37, 437, 500, 819, 836, 850, 904, 1043, 1115, 1200, 1208, 4932, 4946, 5210-5211, 8229, 13488, 17584, 25480, 25619, 28709
1115	37, 367, 437, 500, 836, 903, 1114, 1200, 1208, 4932, 5210-5211, 8229, 13488, 17584, 25479, 28709
1122	37, 256, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 775, 819, 833, 836, 838, 850, 870-871, 875, 880, 905, 921-922, 1025-1027, 1097, 1112, 1200, 1208, 1252, 1257, 4386, 4929, 4932, 4934, 4946, 4971, 5123, 5353, 8229, 8482, 8612, 9025, 9030, 13121, 13488, 16804, 17584, 28709
1123	819, 1124-1125, 1148, 1200, 1208, 1251-1252, 1283, 5347, 13488, 17584
1124	37, 500, 1123, 1125, 1200, 1208, 1251, 1283, 5347, 8229, 13488, 17584, 28709
1125	500, 1123, 1124, 1200, 1208, 1251, 1283, 5347, 13488, 17584
1126	37, 367, 437, 500, 819, 833, 850, 1088, 1200, 1208, 1252, 4929, 4946, 8229, 9025, 13121, 13488, 17584, 25664, 28709
1127	420, 864, 1046, 1089, 1256, 4960, 5142, 8612, 9056, 9238, 16804, 17248
1129	500, 1130, 1200, 1208, 1258, 5354, 13488, 17584
1130	37, 500, 819, 850, 1129, 1200, 1208, 1252, 1258, 4946, 5354, 8229, 13488, 17584, 28709
1131	37, 500, 878, 915, 1025, 1200, 1208, 1251, 1283, 5347, 8229, 13488, 17584, 28709
1132	37, 500, 819, 850, 1133, 1200, 1208, 1252, 4946, 8229, 13488, 17584, 28709
1133	500, 1132, 1200, 1208, 13488, 17584
1137	37, 500, 819, 1200, 1208, 8229, 13488, 17584, 28709
1139	290, 1027, 4386, 5123, 8482
1140	37, 273, 277-278, 280, 284-285, 297, 437, 500, 808, 819, 850, 858, 860, 863, 871-872, 901-902, 923-924, 1013, 1047, 1051, 1141-1149, 1153-1157, 1160-1162, 1164, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1141	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 871-872, 923-924, 1047, 1051, 1140, 1142-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1142	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 865, 871-872, 923-924, 1047, 1051, 1140-1141, 1143-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1143	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 865, 871-872, 923-924, 1047, 1051, 1140-1142, 1144-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1144	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 871-872, 923-924, 1047, 1051, 1140-1143, 1145-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1145	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 860, 863, 871-872, 923-924, 1047, 1051, 1140-1144, 1146-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709

Table 107. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
1146	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 860, 863, 871-872, 923-924, 1047, 1051, 1140-1145, 1147-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1147	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 871-872, 923-924, 1047, 1051, 1140-1146, 1148-1149, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1148	37, 273, 277-278, 280, 284-285, 297, 437, 500, 808, 819, 848-850, 858, 860-861, 863, 865, 871-872, 901-902, 923-924, 1047, 1051, 1123, 1140-1147, 1149, 1153-1164, 1200, 1208, 1252, 1275, 4899, 4946, 5348, 5349, 8229, 12712, 13488, 17584, 28709
1149	37, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 861, 863, 871-872, 923-924, 1047, 1051, 1140-1148, 1153-1157, 1160-1162, 1200, 1208, 1252, 1275, 4946, 5348, 8229, 13488, 17584, 28709
1153	808, 858-859, 867, 872, 923-924, 1140-1149, 1154-1157, 1160-1162, 1200, 1208, 5348, 9044, 13488, 17584
1154	808, 849, 858-859, 867, 872, 923-924, 1140-1149, 1153, 1155-1157, 1160-1162, 1200, 1208, 5347, 5348, 13488, 17584
1155	858-859, 867, 872, 923-924, 1140-1149, 1153-1154, 1156-1157, 1160-1162, 1200, 1208, 5348, 5350, 9049, 13488, 17584
1156	858-859, 901-902, 923-924, 1140-1149, 1153-1155, 1157, 1160, 1200, 1208, 5348, 5353, 12712, 13488, 17584
1157	858-859, 901-902, 923-924, 1140-1149, 1153-1156, 1160, 1200, 1208, 5348, 5353, 12712, 13488, 17584
1158	848, 923, 1148, 1200, 1208, 5347, 5348, 13488, 17584
1159	1148, 1200, 1208, 13488, 17584
1160	858-859, 867, 923-924, 1140-1149, 1153-1157, 1161-1162, 1200, 1208, 5348, 13488, 17584
1161	259, 858-859, 923-924, 1140-1149, 1153-1155, 1160, 5348, 17584
1162	259, 858-859, 923-924, 1140-1149, 1153-1155, 1160, 5348, 17584
1163	924, 1148, 1164, 5354, 17584
1164	858-859, 923-924, 1140, 1148, 1163, 1200, 1208, 5348, 5354, 13488, 17584
1200	37, 256, 259, 273, 275, 277-278, 280, 282, 284-285, 290, 293, 297, 300-301, 367, 420, 423-424, 437, 500, 720, 737, 775, 803, 806, 808, 813, 819, 833-838, 848-852, 855-872, 874-875, 878, 880, 891, 895-897, 901-905, 912, 914-916, 918, 920-924, 927-928, 930, 932-933, 935, 937, 939, 941-944, 946-951, 1004, 1006, 1008-1019, 1025-1027, 1040-1043, 1046-1047, 1051, 1088-1089, 1097-1098, 1112, 1114-1115, 1122-1126, 1129-1133, 1137, 1140-1149, 1153-1160, 1164, 1200, 1250-1258, 1275-1277, 1280-1285, 1351, 1362-1364, 1370-1371, 1380-1381, 1385-1386, 1388, 1390, 1399, 4899, 4909, 4930, 4933, 4948, 4951-4952, 4960, 4971, 5012, 5039, 5104, 5123, 5142, 5210, 5346-5354, 8482, 8612, 9027, 9030, 9044, 9048-9049, 9056, 9061, 9066, 9238, 12712, 13121, 13218, 13488, 16684, 16804, 17248, 17584, 21427, 28709
1208	37, 256, 259, 273, 275, 277-278, 280, 282, 284-285, 290, 293, 297, 300-301, 367, 420, 423-424, 437, 500, 720, 737, 775, 803, 806, 808, 813, 819, 833-838, 848-852, 855-872, 874-875, 878, 880, 891, 895-897, 901-905, 912, 914-916, 918, 920-924, 927-928, 930, 932-933, 935, 937, 939, 941-944, 946-951, 1004, 1006, 1008-1019, 1025-1027, 1040-1043, 1046-1047, 1051, 1088-1089, 1097-1098, 1112, 1114-1115, 1122-1126, 1129-1133, 1137, 1140-1149, 1153-1160, 1164, 1200, 1250-1258, 1275-1277, 1280-1285, 1351, 1362-1364, 1370-1371, 1380-1381, 1385-1386, 1388, 1390, 1399, 4899, 4909, 4930, 4933, 4948, 4951-4952, 4960, 4971, 5012, 5039, 5104, 5123, 5142, 5210, 5346-5354, 8482, 8612, 9027, 9030, 9044, 9048-9049, 9056, 9061, 9066, 9238, 12712, 13121, 13218, 13488, 16684, 16804, 17248, 17584, 21427, 28709

Table 107. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
1250	37, 259, 273, 500, 819, 850, 852, 855, 870, 912, 1200, 1208, 1252, 1282, 4946, 4948, 4951, 5346, 8229, 9044, 13488, 17584, 28709
1251	37, 256, 259, 500, 819, 850, 855, 866, 878, 880, 915, 1025, 1123-1125, 1131, 1200, 1208, 1252, 1283, 4946, 4951, 5347, 8229, 13488, 17584, 28709
1252	37, 256, 259, 273, 275, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 737, 775, 803, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-866, 869-871, 874-875, 880, 897, 903, 905, 912, 914-916, 920-924, 1025-1027, 1041, 1047, 1051, 1097-1098, 1112, 1122-1123, 1126, 1130, 1132, 1140-1149, 1200, 1208, 1250-1251, 1254-1255, 1257, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 16804, 17248, 17584, 25473, 25479, 25617, 28709
1253	37, 259, 423, 500, 737, 813, 819, 850, 869, 875, 1200, 1208, 1280, 4909, 4946, 4971, 5349, 8229, 9061, 13488, 17584, 28709
1254	37, 259, 500, 819, 850, 857, 869, 905, 920, 1026, 1047, 1200, 1208, 1252, 1281, 4946, 4953, 5350, 8229, 9049, 9061, 13488, 17584, 28709
1255	37, 259, 424, 500, 803, 819, 850, 856, 862, 916, 1200, 1208, 1252, 1281, 4946, 4952, 5012, 5351, 8229, 13488, 17584, 28709
1256	259, 420, 500, 720, 850, 864, 1046, 1089, 1127, 1200, 1208, 4946, 4960, 5142, 5352, 8612, 9056, 9238, 13488, 16804, 17248, 17584
1257	37, 259, 437, 500, 775, 819, 850, 914, 921-922, 1112, 1122, 1200, 1208, 1252, 4946, 5353, 8229, 13488, 17584, 28709
1258	37, 259, 500, 819, 1129-1130, 1200, 1208, 5354, 8229, 13488, 17584, 28709
1275	37, 256, 273, 277-278, 280, 284-285, 297, 437, 500, 819, 850, 858, 863, 871, 923-924, 1051, 1140-1149, 1200, 1208, 1252, 4946, 5348, 8229, 13488, 17584, 28709
1276	1200, 1208, 13488, 17584
1277	1200, 1208, 13488, 17584
1280	37, 423, 437, 500, 737, 813, 819, 850, 869, 875, 1200, 1208, 1252-1253, 4909, 4946, 4971, 5349, 8229, 9061, 13488, 17584, 28709
1281	37, 437, 500, 819, 850, 857, 905, 920, 1026, 1200, 1208, 1252, 1254-1255, 4946, 4953, 5350, 8229, 9049, 13488, 17584, 28709
1282	500, 852, 870, 912, 1200, 1208, 1250, 4948, 5346, 9044, 13488, 17584
1283	37, 437, 500, 819, 850, 855, 866, 878, 880, 915, 1025, 1123-1125, 1131, 1200, 1208, 1251-1252, 4946, 4951, 5347, 8229, 13488, 17584, 28709
1284	1200, 1208, 13488, 17584
1285	1200, 1208, 13488, 17584
1351	300-301, 941, 1200, 1208, 4396, 8492, 13488, 16684, 17584
1362	834, 951, 1200, 1208, 4930, 9026, 13488, 17584
1363	933, 949, 1200, 1208, 1364, 5029, 5045, 5460, 9125, 9555, 13221, 13488, 13651, 17317, 17584, 25525, 29621, 33717, 37813
1364	933, 949, 1200, 1208, 1363, 5029, 5045, 5460, 9125, 9555, 13221, 13488, 13651, 17317, 17584, 25525, 29621, 33717, 37813
1370	937-938, 948, 950, 1200, 1208, 1371, 5033, 5046, 9142, 13488, 17584, 25514, 25524, 29620
1371	1200, 1208, 1370, 13488, 17584
1380	837, 928, 1200, 1208, 1385, 4933, 13488, 17584

Table 107. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
1381	935-936, 1200, 1208, 1386, 1388, 5031, 5477, 5482, 5484, 9127, 13223, 13488, 17584, 25512
1385	837, 1200, 1208, 1380, 4933, 13488, 17584
1386	935, 1200, 1208, 1381, 1388, 5031, 5477, 5482, 5484, 9127, 13223, 13488, 17584
1388	935, 1200, 1208, 1381, 1386, 5031, 5477, 5482, 5484, 5488, 9127, 13223, 13488, 17584
1390	930-932, 939, 942-943, 1200, 1208, 1399, 5026, 5028, 5035, 5038-5039, 5055, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
1399	930-932, 939, 942-943, 1200, 1208, 1390, 5026, 5028, 5035, 5038-5039, 5050, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
4386	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 860-865, 870-871, 895-897, 1009, 1025-1027, 1040-1043, 1088, 1112, 1122, 1139, 1252, 4929, 4932, 4946, 4948, 4951, 4953, 4960, 4992, 5123, 8229, 8482, 9025, 9044, 9049, 9056, 13121, 17248, 25473, 25617, 25619, 25664, 28709
4396	300-301, 941, 1351, 8492, 16684
4899	867, 1148, 1200, 1208, 5351, 9048, 12712, 13488, 17584
4909	37, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252-1253, 1280, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5349, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
4929	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 860-865, 870-871, 891, 1009, 1025-1027, 1040-1043, 1088, 1112, 1122, 1126, 1252, 4386, 4932, 4946, 4948, 4951, 4953, 4960, 5123, 8229, 8482, 9025, 9044, 9049, 9056, 13121, 17248, 25617, 25619, 25664, 28709
4930	834, 951, 1200, 1208, 1362, 9026, 13488, 17584
4931	835, 927, 947, 9027, 21427
4932	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 424, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 870-871, 875, 903, 1009, 1025-1027, 1040-1043, 1088, 1112, 1114-1115, 1122, 1252, 4386, 4929, 4946, 4948, 4951, 4953, 4971, 5123, 5210-5211, 8229, 8482, 9025, 9044, 9049, 13121, 25479, 25617, 25619, 25664, 28709
4933	837, 1200, 1208, 1380, 1385, 13488, 17584
4934	37, 256, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 775, 813, 819, 838, 850, 852, 857, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1112, 1122, 1252, 4909, 4946, 4948, 4953, 4960, 4970-4971, 5012, 5123, 8229, 9030, 9044, 9049, 9056, 9061, 9066, 17248, 25473, 25479, 25617, 25619, 28709
4946	37, 256, 259, 273, 275, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 737, 775, 803, 813, 819, 833, 836, 838, 850, 852, 855-858, 860-866, 869-871, 874-875, 880, 897, 903, 905, 912, 914-916, 920-924, 1004, 1025-1027, 1040-1043, 1047, 1051, 1088-1089, 1097-1098, 1100, 1112, 1114, 1122, 1126, 1130, 1132, 1140-1149, 1250-1257, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4948, 4951-4953, 4960, 4970-4971, 5012, 5123, 5210, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 16804, 17248, 25473, 25479, 25617, 25619, 25664, 28709

Table 107. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
4948	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1040-1043, 1047, 1088, 1097, 1200, 1208, 1250, 1252, 1282, 4386, 4909, 4929, 4932, 4934, 4946, 4951, 4953, 4970-4971, 5012, 5123, 5346, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 25664, 28709
4951	37, 259, 273, 277-278, 280, 284-285, 290, 297, 437, 500, 819, 833, 836, 850, 852, 855, 857, 866, 870-871, 878, 880, 912, 915, 1025-1027, 1040-1043, 1088, 1200, 1208, 1250-1252, 1283, 4386, 4929, 4932, 4946, 4948, 4953, 5123, 5346, 5347, 8229, 8482, 9025, 9044, 9049, 13121, 13488, 17584, 25617, 25619, 25664, 28709
4952	259, 273, 424, 500, 803, 850, 856, 862, 916, 1200, 1208, 1255, 4946, 5012, 5351, 13488, 17584
4953	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1040-1043, 1088, 1097, 1252, 1254, 1281, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4970-4971, 5012, 5123, 5350, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 16804, 25473, 25479, 25617, 25619, 25664, 28709
4960	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 720, 819, 833, 838, 850, 864, 870-871, 875, 880, 905, 918, 1008, 1025-1027, 1046, 1089, 1097, 1127, 1200, 1208, 1252, 1256, 4386, 4929, 4934, 4946, 4971, 5104, 5123, 5142, 5352, 8229, 8482, 8612, 9025, 9030, 9056, 9238, 13121, 13488, 16804, 17248, 17584, 28709
4970	37, 259, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1252, 4909, 4934, 4946, 4948, 4953, 4971, 5012, 5123, 8229, 9030, 9044, 9049, 9061, 9066, 25473, 25479, 25617, 25619, 28709
4971	37, 256, 273, 277-278, 280, 284-285, 297, 367, 423, 437, 500, 737, 775, 813, 819, 836, 838, 850-852, 857, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1009, 1025-1027, 1041-1043, 1047, 1088, 1112, 1122, 1200, 1208, 1252-1253, 1280, 4909, 4932, 4934, 4946, 4948, 4953, 4960, 4970, 5012, 5123, 5349, 8229, 9030, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
4992	290, 896, 1027, 1041, 4386, 5123, 8482, 25617
5012	37, 273, 277-278, 280, 284-285, 297, 423-424, 437, 500, 803, 813, 819, 838, 850, 852, 856-857, 860-863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252, 1255, 4909, 4934, 4946, 4948, 4952-4953, 4970-4971, 5123, 5351, 8229, 9030, 9044, 9049, 9061, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
5026	930-932, 939, 942-943, 1390, 1399, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
5028	930-932, 939, 942-943, 1390, 1399, 5026, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
5029	933-934, 944, 949, 1363-1364, 5045, 5460, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 33717, 37813
5031	935-936, 946, 1381, 1386, 1388, 5477, 5482, 5484, 9127, 13223, 25512
5033	937-938, 948, 950, 1370, 5046, 9142, 25514, 25524, 29620
5035	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
5038	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796

Table 107. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
5039	930-932, 939, 942-943, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038, 9122, 9124, 9131, 9135, 13218-13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
5045	933-934, 944, 949, 1363-1364, 5029, 5460, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 33717, 37813
5046	937-938, 948, 950, 1370, 5033, 9142, 25514, 25524, 29620
5104	420, 864, 1008, 1200, 1208, 4960, 8612, 9056, 13488, 16804, 17248, 17584
5123	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 423, 437, 500, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-865, 869-871, 874-875, 880, 895-897, 903, 912, 916, 1025-1027, 1040-1043, 1047, 1088, 1112, 1122, 1139, 1200, 1208, 1252, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 4992, 5012, 8229, 8482, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 25664, 28709
5142	420, 500, 864, 1046, 1089, 1127, 1200, 1208, 1256, 4960, 5352, 8612, 9056, 9238, 13488, 16804, 17248, 17584
5210	37, 437, 500, 819, 836, 850, 904, 1043, 1114-1115, 1200, 1208, 4932, 4946, 5211, 8229, 13488, 17584, 25480, 25619, 28709
5211	37, 367, 437, 500, 836, 903, 1114-1115, 4932, 5210, 8229, 25479, 28709
5346	37, 259, 273, 500, 819, 850, 852, 855, 870, 912, 1200, 1208, 1250, 1252, 1282, 4946, 4948, 4951, 8229, 9044, 13488, 17584, 28709
5347	808, 848-849, 855, 866, 872, 878, 880, 915, 1025, 1123-1125, 1131, 1154, 1158, 1200, 1208, 1251, 1283, 4951, 13488, 17584
5348	37, 259, 273, 275, 277-278, 280, 284-285, 297, 437, 500, 808, 819, 850, 858, 860-861, 863, 865, 871-872, 901-902, 923-924, 1051, 1140-1149, 1153-1158, 1160-1162, 1164, 1200, 1208, 1252, 1275, 4946, 8229, 13488, 17584, 28709
5349	813, 869, 875, 1148, 1200, 1208, 1253, 1280, 4909, 4971, 9061, 13488, 17584
5350	857, 920, 1026, 1155, 1200, 1208, 1254, 1281, 4953, 9049, 13488, 17584
5351	424, 856, 862, 867, 916, 1200, 1208, 1255, 4899, 4952, 5012, 9048, 12712, 13488, 17584
5352	420, 864, 1046, 1089, 1200, 1208, 1256, 4960, 5142, 8612, 9056, 9238, 13488, 16804, 17248, 17584
5353	901-902, 921-922, 1112, 1122, 1156-1157, 1200, 1208, 1257, 13488, 17584
5354	1129-1130, 1163, 1164, 1200, 1208, 1258, 13488, 17584
5460	933-934, 944, 949, 1363-1364, 5029, 5045, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 33717, 37813
5477	935-936, 1381, 1386, 1388, 5031, 5482, 5484, 9127, 13223, 25512
5482	935, 1381, 1386, 1388, 5031, 5477, 5484, 9127, 13223
5484	935-936, 946, 1381, 1386, 1388, 5031, 5477, 5482, 9127, 13223, 25512
5488	1388
8229	37, 256, 273, 275, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 720, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-866, 869-871, 874-875, 880, 897, 903-905, 912, 914-916, 920-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1097, 1100, 1112, 1114-1115, 1122, 1124, 1126, 1130-1132, 1137, 1140-1149, 1250-1255, 1257-1258, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5210-5211, 5346, 5348, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 16804, 17248, 25473, 25479, 25480, 25617, 25619, 25664, 28709

Table 107. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
8482	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 860-865, 870-871, 895-897, 1009, 1025-1027, 1040-1043, 1088, 1112, 1122, 1139, 1200, 1208, 1252, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4960, 4992, 5123, 8229, 9025, 9044, 9049, 9056, 13121, 13488, 17248, 17584, 25473, 25617, 25619, 25664, 28709
8492	300-301, 941, 1351, 4396, 16684
8612	37, 256, 420, 424, 437, 500, 720, 737, 775, 819, 850, 852, 857, 860-865, 1008, 1046, 1089, 1098, 1112, 1122, 1127, 1200, 1208, 1252, 1256, 4946, 4948, 4953, 4960, 5104, 5142, 5352, 8229, 9044, 9049, 9056, 9238, 13488, 16804, 17248, 17584, 28709
9025	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 860-865, 870-871, 891, 1009, 1025-1027, 1040-1043, 1088, 1112, 1122, 1126, 1252, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4960, 5123, 8229, 8482, 9044, 9049, 9056, 13121, 17248, 25617, 25619, 25664, 28709
9026	834, 926, 951, 1362, 4930
9027	835, 927, 947, 1200, 1208, 4931, 13488, 17584, 21427
9030	37, 256, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 775, 813, 819, 838, 850, 852, 857, 860-865, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1112, 1122, 1200, 1208, 1252, 4909, 4934, 4946, 4948, 4953, 4960, 4970-4971, 5012, 5123, 8229, 9044, 9049, 9056, 9061, 9066, 13488, 17248, 17584, 25473, 25479, 25617, 25619, 28709
9044	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1040-1043, 1047, 1088, 1097, 1153, 1200, 1208, 1250, 1252, 1282, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5346, 8229, 8482, 8612, 9025, 9030, 9049, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 25664, 28709
9048	867, 1200, 1208, 4899, 5351, 12712, 13488, 17584
9049	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 905, 912, 916, 920, 1025-1027, 1040-1043, 1088, 1097, 1155, 1200, 1208, 1252, 1254, 1281, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5350, 8229, 8482, 8612, 9025, 9030, 9044, 9061, 9066, 13121, 13488, 16804, 17584, 25473, 25479, 25617, 25619, 25664, 28709
9056	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 720, 819, 833, 838, 850, 864, 870-871, 875, 880, 905, 918, 1008, 1025-1027, 1046, 1089, 1097, 1127, 1200, 1208, 1252, 1256, 4386, 4929, 4934, 4946, 4960, 4971, 5104, 5123, 5142, 5352, 8229, 8482, 8612, 9025, 9030, 9238, 13121, 13488, 16804, 17248, 17584, 28709
9061	37, 256, 259, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 737, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252-1254, 1280, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5349, 8229, 9030, 9044, 9049, 9066, 13488, 17584, 25473, 25479, 25617, 25619, 28709
9066	37, 259, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1200, 1208, 1252, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 8229, 9030, 9044, 9049, 9061, 13488, 17584, 25473, 25479, 25617, 25619, 28709
9122	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
9124	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796

Table 107. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
9125	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 33717, 37813
9127	935-936, 946, 1381, 1386, 1388, 5031, 5477, 5482, 5484, 13223, 25512
9131	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
9135	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
9142	937-938, 948, 950, 1370, 5033, 5046, 25514, 25524, 29620
9238	420, 500, 864, 1046, 1089, 1127, 1200, 1208, 1256, 4960, 5142, 5352, 8612, 9056, 13488, 16804, 17248, 17584
9555	933, 949, 1363-1364, 5029, 5045, 5460, 9125, 13221, 13651, 17317, 25525, 29621, 33717, 37813
12712	862, 867, 1148, 1156-1157, 1200, 1208, 4899, 5351, 9048, 13488, 17584
13121	37, 256, 273, 277-278, 280, 284-285, 290, 297, 367, 437, 500, 737, 775, 819, 833, 836, 850, 852, 855, 857, 860-865, 870-871, 891, 1009, 1025-1027, 1040-1043, 1088, 1112, 1122, 1126, 1200, 1208, 1252, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4960, 5123, 8229, 8482, 9025, 9044, 9049, 9056, 13488, 17248, 17584, 25617, 25619, 25664, 28709
13218	930-932, 939, 942-943, 1200, 1208, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13219, 13231, 13488, 17314, 17584, 25508, 25518, 29614, 33698-33700, 37796
13219	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218, 13231, 17314, 25508, 25518, 29614, 33698-33700, 37796
13221	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 33717, 37813
13223	935-936, 946, 1381, 1386, 1388, 5031, 5477, 5482, 5484, 9127, 25512
13231	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 17314, 25508, 25518, 29614, 33698-33700, 37796
13488	37, 256, 259, 273, 275, 277-278, 280, 282, 284-285, 290, 293, 297, 300-301, 367, 420, 423-424, 437, 500, 720, 737, 775, 803, 806, 808, 813, 819, 833-838, 848-852, 855-872, 874-875, 878, 880, 891, 895-897, 901-905, 912, 914-916, 918, 920-924, 927-928, 930, 932-933, 935, 937, 939, 941-944, 946-951, 1004, 1006, 1008-1019, 1025-1027, 1040-1043, 1046-1047, 1051, 1088-1089, 1097-1098, 1112, 1114-1115, 1122-1126, 1129-1133, 1137, 1140-1149, 1153-1160, 1164, 1200, 1208, 1250-1258, 1275-1277, 1280-1285, 1351, 1362-1364, 1370-1371, 1380-1381, 1385-1386, 1388, 1390, 1399, 4899, 4909, 4930, 4933, 4948, 4951-4952, 4960, 4971, 5012, 5039, 5104, 5123, 5142, 5210, 5346-5354, 8482, 8612, 9027, 9030, 9044, 9048-9049, 9056, 9061, 9066, 9238, 12712, 13121, 13218, 16684, 16804, 17248, 17584, 21427, 28709
13651	933, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 17317, 25525, 29621, 33717, 37813
16684	300-301, 941, 1200, 1208, 1351, 4396, 8492, 13488, 17584
16804	37, 256, 420, 424, 437, 500, 720, 737, 775, 819, 850, 852, 857, 860-865, 1008, 1046, 1089, 1098, 1112, 1122, 1127, 1200, 1208, 1252, 1256, 4946, 4948, 4953, 4960, 5104, 5142, 5352, 8229, 8612, 9044, 9049, 9056, 9238, 13488, 17248, 17584, 28709
17248	37, 256, 259, 273, 277-278, 280, 284-285, 290, 297, 420, 423-424, 500, 720, 819, 833, 838, 850, 864, 870-871, 875, 880, 905, 918, 1008, 1025-1027, 1046, 1089, 1097, 1127, 1200, 1208, 1252, 1256, 4386, 4929, 4934, 4946, 4960, 4971, 5104, 5123, 5142, 5352, 8229, 8482, 8612, 9025, 9030, 9056, 9238, 13121, 13488, 16804, 17584, 28709

Table 107. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
17314	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 25508, 25518, 29614, 33698-33700, 37796
17317	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13651, 25510, 25520, 25525, 29616, 29621, 33717, 37813
17584	37, 256, 259, 273, 275, 277-278, 280, 282, 284-285, 290, 293, 297, 300-301, 367, 420, 423-424, 437, 500, 720, 737, 775, 803, 806, 808, 813, 819, 833-838, 848-852, 855-872, 874-875, 878, 880, 891, 895-897, 901-905, 912, 914-916, 918, 920-924, 927-928, 930, 932-933, 935, 937, 939, 941-944, 946-951, 1004, 1006, 1008-1019, 1025-1027, 1040-1043, 1046-1047, 1051, 1088-1089, 1097-1098, 1112, 1114-1115, 1122-1126, 1129-1133, 1137, 1140-1149, 1153-1160, 1164, 1200, 1208, 1250-1258, 1275-1277, 1280-1285, 1351, 1362-1364, 1370-1371, 1380-1381, 1385-1386, 1388, 1390, 1399, 4899, 4909, 4930, 4933, 4948, 4951-4952, 4960, 4971, 5012, 5039, 5104, 5123, 5142, 5210, 5346-5354, 8482, 8612, 9027, 9030, 9044, 9048-9049, 9056, 9061, 9066, 9238, 12712, 13121, 13218, 13488, 16684, 16804, 17248, 21427, 28709
21427	835, 927, 947, 1200, 1208, 4931, 9027, 13488, 17584
25473	37, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 813, 819, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1252, 4386, 4909, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 8229, 8482, 9030, 9044, 9049, 9061, 9066, 25479, 25617, 25619, 28709
25479	37, 273, 277-278, 280, 284-285, 297, 423, 437, 500, 813, 819, 836, 838, 850, 852, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 920, 1025-1027, 1041-1043, 1115, 1252, 4909, 4932, 4934, 4946, 4948, 4953, 4970-4971, 5012, 5123, 5211, 8229, 9030, 9044, 9049, 9061, 9066, 25473, 25617, 25619, 28709
25480	37, 500, 904, 1114, 5210, 8229, 28709
25508	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25518, 29614, 33698-33700, 37796
25510	933-934, 949, 5029, 5045, 5460, 9125, 13221, 17317, 25525, 29621, 33717, 37813
25512	935-936, 946, 1381, 5031, 5477, 5484, 9127, 13223
25514	937-938, 950, 1370, 5033, 5046, 9142
25518	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 29614, 33698-33700, 37796
25520	933, 944, 949, 5029, 5045, 5460, 9125, 13221, 17317, 25525, 29616, 29621, 33717, 37813
25524	937, 948, 950, 1370, 5033, 5046, 9142, 29620
25525	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 29616, 29621, 33717, 37813
25617	37, 273, 277-278, 280, 284-285, 290, 297, 367, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 895-897, 903, 912, 916, 1025-1027, 1040-1043, 1088, 1252, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 4992, 5012, 5123, 8229, 8482, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 25473, 25479, 25619, 25664, 28709
25619	37, 273, 277-278, 280, 284-285, 290, 297, 423, 437, 500, 813, 819, 833, 836, 838, 850, 852, 855, 857, 860-861, 863, 869-871, 874-875, 880, 897, 903, 912, 916, 1025-1027, 1040-1043, 1088, 1114, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4970-4971, 5012, 5123, 5210, 8229, 8482, 9025, 9030, 9044, 9049, 9061, 9066, 13121, 25473, 25479, 25617, 25664, 28709
25664	37, 273, 277-278, 280, 284-285, 290, 297, 367, 500, 819, 833, 836, 850, 852, 855, 857, 870-871, 875, 891, 1025-1027, 1040-1043, 1088, 1126, 4386, 4929, 4932, 4946, 4948, 4951, 4953, 4971, 5123, 8229, 8482, 9025, 9044, 9049, 13121, 25617, 25619, 28709

Table 107. WebSphere MQ for z/OS CCSID conversion support (continued)

CCSID	Converts to and from CCSIDS
28709	37, 256, 273, 275, 277-278, 280, 284-285, 290, 297, 367, 420, 423-424, 437, 500, 720, 737, 775, 813, 819, 833, 836, 838, 850, 852, 855, 857-858, 860-866, 869-871, 874-875, 880, 897, 903-905, 912, 914-916, 920-924, 1009, 1025-1027, 1040-1043, 1047, 1051, 1088, 1097, 1100, 1112, 1114-1115, 1122, 1124, 1126, 1130-1132, 1137, 1140-1149, 1200, 1208, 1250-1255, 1257-1258, 1275, 1280-1281, 1283, 4386, 4909, 4929, 4932, 4934, 4946, 4948, 4951, 4953, 4960, 4970-4971, 5012, 5123, 5210-5211, 5346, 5348, 8229, 8482, 8612, 9025, 9030, 9044, 9049, 9056, 9061, 9066, 13121, 13488, 16804, 17248, 17584, 25473, 25479, 25480, 25617, 25619, 25664
29614	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 33698-33700, 37796
29616	933, 944, 949, 5029, 5045, 5460, 9125, 13221, 17317, 25520, 25525, 29621, 33717, 37813
29620	937, 948, 950, 1370, 5033, 5046, 9142, 25524
29621	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 33717, 37813
33698	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33699-33700, 37796
33699	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698, 33700, 37796
33700	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33699, 37796
33717	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 37813
37796	930-932, 939, 942-943, 1390, 1399, 5026, 5028, 5035, 5038-5039, 9122, 9124, 9131, 9135, 13218-13219, 13231, 17314, 25508, 25518, 29614, 33698-33700
37813	933-934, 944, 949, 1363-1364, 5029, 5045, 5460, 9125, 9555, 13221, 13651, 17317, 25510, 25520, 25525, 29616, 29621, 33717

i5/OS conversion support

A full list of CCSIDs, and conversions supported by i5/OS, can be found in the appropriate i5/OS publication relating to your operating system.

Unicode conversion support

Some platforms support the conversion of user data to or from Unicode encoding. The two forms of Unicode encoding supported are UCS-2 (CCSIDs 1200, 13488, and 17584) and UTF-8 (CCSID 1208).

The term *UCS-2* is often used interchangeably but incorrectly with *UTF-16*. *UCS-2* is a fixed-width encoding where each character occupies 2 bytes. *UTF-16* is variable-width encoding that is a superset of *UCS-2*. In addition to the 2-byte *UCS-2* characters, *UTF-16* contains characters, known as surrogate pairs, that are 4 bytes in length. WebSphere MQ does not support surrogate pairs. The support for *UTF-16* and *UTF-8* in WebSphere MQ is therefore limited to those Unicode characters that can be encoded in *UCS-2*.

Note: WebSphere MQ does not support *UCS-2* queue manager CCSIDs so message header data cannot be encoded in *UCS-2*.

WebSphere MQ AIX support for Unicode

On WebSphere MQ for AIX conversion to and from Unicode CCSIDs is supported for the CCSIDs in the following table.

037	273	278	280	284	285
297	423	437	500	813	819
850	852	856	857	858	860
861	865	867	869	875	878
880	901	902	912	915	916
920	923	924	932	933	935
937	938	939	942	943	948
949	950	954	964	970	1026
1046	1089	1129	1130	1131	1132
1133	1140	1141	1142	1143	1144
1145	1146	1147	1148	1149	1200
1153	1156	1157	1208	1250	1251
1253	1254	1258	1280	1281	1282
1283	1284	1285	1363	1364	1381
1383	1386	1388	4899	5026	5035
5050	5346	5347	5348	5349	5350
5351	5352	5353	5354	5488	9044
9048	9449	12712	13488	17584	33722

WebSphere MQ HP-UX support for Unicode

On WebSphere MQ for HP-UX conversion to and from Unicode CCSIDs is supported for the CCSIDs listed in the following table.

437	737	813	819	850	852
855	857	861	864	865	866
869	874	912	915	916	920
932	938	950	954	964	970
1051	1089	1140	1141	1142	1143
1144	1145	1146	1147	1148	1149
1200	1208	1250	1251	1252	1253
1254	1255	1256	1257	1258	1381
5050	5488	13488	33722		

WebSphere MQ for Windows, Solaris, and Linux support for Unicode

On WebSphere MQ for Windows, WebSphere MQ for Solaris, and WebSphere MQ for Linux conversion to, and from, Unicode CCSIDs is supported for the CCSIDs in the following table.

037	277	278	280	284	285
290	297	300	301	420	424
437	500	813	819	833	835
836	837	838	850	852	855
856	857	858	860	861	862
863	864	865	866	867	868
869	870	871	874	875	878
880	891	897	901	902	903
904	912	913 (5)	915	916	918

920	921	922	923	924	927
928	930	931 (1)	932 (2)	933	935
937	938 (3)	939	941	942	943
947	948	949	950	951	954 (4)
964	970	1006	1025	1026	1027
1040	1041	1042	1043	1046	1047
1051	1088	1089	1097	1098	1112
1114	1115	1122	1123	1124	1129
1130	1132	1133	1140	1141	1142
1143	1144	1145	1146	1147	1148
1149	1153	1156	1157	1200	1208
1250	1251	1252	1253	1254	1255
1256	1257	1258	1275	1280	1281
1282	1283	1363	1364	1380	1381
1383	1386	1388	4899	5050	5346
5347	5348	5349	5350	5351	5352
5353	5354	5488 (5)	9044	9048	9449
12712	13488	17584	33722 (4)		

Notes:

1. – 931 uses 939 for conversion.
2. – 932 uses 942 for conversion.
3. – 938 uses 948 for conversion.
4. – 954 and 33722 use 5050 for conversion.
5. – On Windows, Linux, OVMS V5.3, and Solaris only.

i5/OS support for Unicode

For details on UNICODE support refer to the appropriate i5/OS publication relating to your operating system.

WebSphere MQ for z/OS support for Unicode

On WebSphere MQ for z/OS conversion to and from the Unicode CCSIDs is supported for the following CCSIDs:

37	256	259	273	275	277
278	280	282	284	285	290
293	297	300	301	367	420
423	424	437	500	720	737
775	803	806	808	813	819
833	834	835	836	837	838
848	849	850	851	852	855
856	857	858	859	860	861
862	863	864	865	866	867
868	869	870	871	872	874
875	878	880	891	895	896
897	901	902	903	904	905
912	914	915	916	918	920
921	922	923	924	927	928
930	932	933	935	937	939
941	942	943	944	946	947
948	949	950	951	1004	1006
1008	1009	1010	1011	1012	1013
1014	1015	1016	1017	1018	1019

1025	1026	1027	1040	1041	1042
1043	1046	1047	1051	1088	1089
1097	1098	1112	1114	1115	1122
1123	1124	1125	1126	1129	1130
1131	1132	1133	1137	1140	1141
1142	1143	1144	1145	1146	1147
1148	1149	1153	1154	1155	1156
1157	1158	1159	1160	1161	1162
1164	1200	1208	1250	1251	1252
1253	1254	1255	1256	1257	1258
1275	1276	1277	1280	1281	1282
1283	1284	1285	1351	1362	1363
1364	1370	1371	1380	1381	1385
1386	1388	1390	1399	4899	4909
4930	4933	4948	4951	4952	4960
4971	5012	5039	5104	5123	5142
5210	5346	5347	5348	5349	5350
5351	5352	5353	5354	5488	8482
8612	9027	9030	9044	9048	9049
9056	9061	9066	9238	9449	12712
13121	13218	13488	16684	16804	17248
17584	21427	28709			

Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing,
IBM Corporation,
North Castle Drive,
Armonk, NY 10504-1785,
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation,
Licensing,
2-31 Roppongi 3-chome, Minato-k,u
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AD/Cycle
AIX
AS/400
CICS
CICS/VSE
DB2
DB2 Universal Database
FFST
i5/OS
IBM
IBMLink

IMS
Informix
iSeries
Language Environment
Lotus
Lotus Notes
MQSeries
MVS
OS/2
OS/390
OS/400
PowerPC
RACF
S/390
SAA
SupportPac
System/370
System/390
Tivoli
TXSeries
VisualAge
WebSphere
z/OS
zSeries

Lotus[®] and Lotus Notes are registered trademarks of Lotus Development Corporation in the United States, or other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Index

A

- AbendCode field 59
- AccountingConnOverride attribute
 - queue manager 578
- AccountingInterval attribute
 - queue manager 578
- AccountingToken field
 - MQMD structure 183
 - MQPMR structure 301
- Action field
 - MQPMO structure 274
- AdoptNewMCACheck attribute
 - queue manager 579
- AdoptNewMCAType attribute
 - queue manager 579
- ADSDescriptor field 59
- aliasing
 - queue manager 616
 - reply queue 616
- AlterationDate attribute
 - namelist 652
 - process definition 654
 - queue 619
 - queue manager 579
- AlterationTime attribute
 - namelist 652
 - process definition 654
 - queue 620
 - queue manager 580
- AlternateSecurityId field 251
- AlternateUserId field 251
- AppId
 - attribute 655
 - field
 - MQTM structure 368
 - MQTMC2 structure 375
- AppIdentityData field 185
- AppOriginData field 186
- AppType
 - attribute 655
 - field
 - MQTM structure 368
 - MQTMC2 structure 375
- AppOptions field 685
- AttentionId field 60
- attributes
 - namelist 651
 - process definition 654
 - queue 615
 - queue manager 575
- authentication information record 29
- AuthenticationType field
 - MQCSP structure 93
- Authenticator field 60, 164
- AuthInfoConnName field
 - MQAIR structure 29
- AuthInfoRecCount field
 - MQSCO structure 328
- AuthInfoRecOffset field
 - MQSCO structure 328

- AuthInfoRecPtr field
 - MQSCO structure 328
- AuthInfoType field
 - MQAIR structure 30
- AuthorityEvent attribute 580

B

- BackoutCount field 186
- BackoutRequeueQName attribute 620
- BackoutThreshold attribute 620
- BaseQName attribute 621
- BaseType attribute 621
- begin options structure 36
- BeginOptions parameter 402
- BridgeEvent attribute
 - queue manager 580
- Buffer parameter
 - declaring 397
 - MQCB_FUNCTION call 421
 - MQGET call 468
 - MQPUT call 526
 - MQPUT1 call 540
- buffer to message handle options 33
- BufferLength parameter
 - MQGET call 468
 - MQPUT call 525
 - MQPUT1 call 540
- built-in formats 197

C

- C programming language
 - data types 21
 - functions 20
 - header files 20
 - initial values for dynamic structures 22
 - initial values for structures 21
 - manipulating binary strings 21
 - manipulating character strings 21
 - notational conventions 23
 - parameters with undefined data types 20
 - use from C++ 22
 - using calls 397
- callback area field
 - MQCBD structure 47
- callback context structure 38
- callback descriptor structure 46
- CallbackArea field
 - MQCBD structure 39
- CallbackDesc parameter
 - MQCB call 411
- CallbackFunction field
 - MQCBD structure 47
- CallbackName field
 - MQCBD structure 47
- CallbackType field
 - MQCBD structure 49

- calls
 - conventions used 395
 - detailed description
 - MQ_DATA_CONV_EXIT 697
 - MQBACK 397
 - MQBEGIN 401
 - MQBUFMFH 405
 - MQCB 409
 - MQCB_FUNCTION 419
 - MQCLOSE 423
 - MQCMIT 431
 - MQCONN 435
 - MQCONNX 444
 - MQCRTMH 447
 - MQCTL 450
 - MQDISC 457
 - MQDLTMH 461
 - MQDLTMP 464
 - MQGET 467
 - MQINQ 480
 - MQINQMP 495
 - MQMHBUF 501
 - MQOPEN 505
 - MQPUT 524
 - MQPUT1 539
 - MQSET 550
 - MQSETMP 556
 - MQSTAT 561
 - MQSUB 564
 - MQSUBRQ 571
 - MQXCNVC 691
- CallType field
 - MQCBC structure 39
- CancelCode field 60
- CCSID language support 713
- CFStrucName attribute 621
- ChannelAutoDef attribute 580
- ChannelAutoDefEvent attribute 581
- ChannelAutoDefExit attribute 581
- ChannelEvent attribute
 - queue manager 581
- ChannelInitiatorControl attribute
 - queue manager 582
- ChannelMonitoring attribute
 - queue manager 582
- ChannelStatistics attribute
 - queue manager 583
- CharAttrLength parameter
 - MQINQ call 489
 - MQSET call 552
- CharAttrs parameter
 - MQINQ call 489
 - MQSET call 552
- ChinitAdapters attribute
 - queue manager 583
- ChinitDispatchers attribute
 - queue manager 584
- ChinitTraceAutoStart attribute
 - queue manager 584
- ChinitTraceTableSize attribute
 - queue manager 584

- ClientConnOffset field 79
 - ClientConnPtr field 79
 - ClusterName attribute 622
 - ClusterNameList attribute 622
 - ClusterSenderMonitoringDefault attribute
 - queue manager 585
 - ClusterSenderStatistics attribute
 - queue manager 585
 - ClusterWorkloadData attribute 586
 - ClusterWorkloadExit attribute 586
 - ClusterWorkloadLength attribute 586
 - CLWLMRChannels attribute
 - queue manager 587
 - CLWLQueuePriority attribute 622
 - CLWLQueueRank attribute 623
 - CLWLUseQ attribute 623
 - queue manager 587
 - COBOL programming language
 - COPY files 23
 - named constants 25
 - notational conventions 25
 - pointer data type 25
 - structures 24
 - code-page conversions 713
 - coded character set identifier 587
 - CodedCharSetId
 - attribute 587
 - field
 - MQCIH structure 60
 - MQDH structure 101
 - MQDLH structure 109
 - MQDXP structure 685
 - MQEPH structure 122
 - MQIIH structure 165
 - MQMD structure 187
 - MQMDE structure 242
 - MQRFH structure 304
 - MQRFH2 structure 308
 - MQRMH structure 317
 - MQWIH structure 379
 - Codepage support 715
 - Arabic 728
 - Cyrillic 723
 - Danish and Norwegian 716
 - Eastern European languages 722
 - Estonian 724
 - Farsi 729
 - Finnish and Swedish 717
 - French 720
 - German 716
 - Greek 726
 - Hebrew 727
 - Icelandic 721
 - Italian 718
 - Japanese Kanji/ Katakana Mixed 735
 - Japanese Kanji/ Latin Mixed 734
 - Japanese Katakana SBCS 732
 - Japanese Latin SBCS 731
 - Korean 737
 - Lao 730
 - Latvian and Lithuanian 725
 - Multilingual 720
 - Portuguese 721
 - Simplified Chinese 737
 - Spanish 719
 - Thai 730
 - Traditional Chinese 738
 - Codepage support (*continued*)
 - Turkish 727
 - UK English / Gaelic 719
 - Ukrainian 726
 - Urdu 729
 - US English 715
 - Vietnamese 730
 - CommandEvent attribute
 - queue manager 587
 - CommandInputQName attribute 588
 - CommandLevel attribute 588
 - CommandServerControl attribute 591
 - CommitMode field 165
 - compatibility mode 442
 - CompCode field
 - MQCBC structure 41
 - MQCIH structure 61
 - MQDXP structure 686
 - MQRRE structure 326
 - MQSTS structure 362
 - CompCode parameter
 - MQBACK call 398, 407, 449, 503
 - MQBEGIN call 402
 - MQCB call 413
 - MQCLOSE call 427
 - MQCMIT call 432
 - MQCONN call 439
 - MQCONNX call 444
 - MQCTL call 453
 - MQDISC call 458
 - MQDLTMH call 462
 - MQDLTMP call 464
 - MQGET call 469
 - MQINQ call 490
 - MQINQMP call 498
 - MQOPEN call 513
 - MQPUT call 526
 - MQPUT1 call 540
 - MQSET call 552
 - MQSETMP call 558
 - MQSTAT call 562
 - MQXCNV call 695
 - completion code 661
 - ConfigurationEvent attribute 591
 - connect options structure 77
 - ConnectionArea field
 - MQCBC structure 41
 - MQCTLO structure 97
 - ConnectionId field 81
 - ConnectOpts parameter 444
 - ConnTag field 82
 - Context field 279
 - Context parameter
 - MQCB_FUNCTION call 421
 - control callback options structure 96
 - ControlOpts parameter
 - MQCTLO call 453
 - conversation sharing 86, 87
 - ConversationalTask field 61
 - conversion of report messages 683
 - conversions, code-page 713
 - COPY files – COBOL programming
 - language 23
 - CorrelId field
 - MQMD structure 188
 - MQPMR structure 301
 - create-message options structure 74
 - CreationDate attribute 623
 - CreationTime attribute 624
 - CryptoHardware field
 - MQSCO structure 329
 - CSPPasswordLength field
 - MQCSP structure 93
 - CSPPasswordOffset field
 - MQCSP structure 93
 - CSPPasswordPtr field
 - MQCSP structure 93
 - CSPUseridLength field
 - MQCSP structure 93
 - CSPUserIdOffset field
 - MQCSP structure 93
 - CSPUserIdPtr field
 - MQCSP structure 94
 - CurrentQDepth attribute 624
 - CursorPosition field 61
- ## D
- data conversion
 - processing conventions 679
 - report messages 683
 - data types, conventions used 19
 - data types, detailed description
 - elementary 7, 8, 9, 10, 11
 - assembler language 16
 - C programming language 11
 - COBOL programming
 - language 14
 - MQBOOL 3
 - MQBYTE 3
 - MQBYTEn 4
 - MQCHAR 4
 - MQCHARn 4
 - MQFLOAT32 4
 - MQFLOAT64 5
 - MQHCONFIG 5
 - MQHCONN 5
 - MQHOBJ 5
 - MQINT16 5
 - MQINT32 5
 - MQINT64 6
 - MQINT8 5
 - MQLONG 6
 - MQPID 6
 - MQPTR 6
 - MQTID 6
 - MQUINT16 6
 - MQUINT32 6
 - MQUINT64 7
 - MQUINT8 6
 - MQULONG 7
 - overview 1
 - PL/I language 15
 - PMQBMHO 7
 - PMQBOOL 7
 - PMQCBC 7
 - PMQCBD 7
 - PMQCHAR 7
 - PMQCHARV 8
 - PMQCMHO 8
 - PMQCSP 8
 - MQCTLO 8
 - PMQDH 8
 - PMQDHO 8

data types, detailed description
(continued)

elementary (continued)

PMQDMHO 8
PMQFLOAT32 8
PMQFLOAT64 8
PMQINT16 9
PMQINT32 9
PMQINT64 9
PMQINT8 9
PMQLONG 9
PMQMD 9
PMQRFH 10
PMQTTID 10
PMQUINT16 10
PMQUINT32 10
PMQUINT64 11
PMQUINT8 10
PMQULONG 11

structure

MQAIR 29
MQBMHO 33
MQBO 36
MQCBC 38
MQCBD 46
MQCHARV 53
MQCIH 57
MQCMHO 74
MQCNO 77
MQCSP 92
MQCTLO 96
MQDH 99
MQDLH 106
MQDMHO 116
MQDMPO 118
MQDXP 684
MQEPH 121
MQGMO 126
MQIIH 163
MQIMPO 171
MQMD 181
MQMDE 239
MQMHBO 247
MQOD 249
MQOR 266
MQPMO 272
MQPMR 300
MQRFH 303
MQRFH2 308
MQRMH 315
MQRR 325
MQSCO 327
MQSMPO 355
MQSTS 361
MQTM 366
MQTMC2 374
MQWIH 378
MQXP 383
MQXQH 387
programming considerations 17
rules 18

DataConvExitParms parameter 698

DataLength

field, MQDXP structure 686
parameter
MQGET call 469
MQXCNVC call 695

DataLength parameter

MQINQMP call 498

DataLogicalLength field 318

DataLogicalOffset field 318

DataLogicalOffset2 field 318

dead-letter header structure 106

DeadLetterQName attribute 592

DefaultPutResponse 624

DefBind attribute 625

DefinitionType attribute 625

DefInputOpenOption attribute 626

DefPersistence attribute 627

DefPResp attribute 629

DefPriority attribute 627

DefReadAhead 628

DefXmitQName attribute 593

delete message handle options

structure 116

delete message property options 118

DestEnvLength field 319

DestEnvOffset field 319

DestNameLength field 319

DestNameOffset field 319

DestQMgrName field 109

DestQName field 109

DistLists attribute 593, 629

distribution header structure 99

distribution lists 593, 629

DNSGroup attribute

queue manager 593

DNSWLM attribute

queue manager 593

dynamic queue 506

DynamicQName field 252

E

embedded PCF header structure 121

Encoding field

MQCIH structure 61

MQDH structure 101

MQDLH structure 109

MQDXP structure 686

MQEPH structure 122

MQIIH structure 165

MQMD structure 190

MQMDE structure 243

MQRFH structure 304

MQRFH2 structure 309

MQRMH structure 319

MQWIH structure 379

using 667

EnvData

attribute 657

field

MQTM structure 370

MQTMC2 structure 375

environment variable –

MQ_CONNECT_TYPE 84

ErrorOffset field 61

exit parameter block 383

ExitCommand field 383

ExitId field 384

ExitOptions field 687

ExitParmCount field 384

ExitReason field 384

ExitResponse field

MQDXP structure 687

MQXP structure 385

ExitUserArea field 385

expired-message processing 594

Expiry field 190

ExpiryInterval attribute 594

F

Facility field 61

FacilityKeepTime field 62

FacilityLike field 62

Feedback field

MQMD structure 193

MQPMR structure 301

FipsRequired field

MQSCO structure 330

Flags field

MQCIH structure 62

MQDH structure 102

MQEPH structure 122

MQIIH structure 165

MQMDE structure 243

MQRFH structure 304

MQRFH2 structure 309

MQRMH structure 320

MQWIH structure 379

Format field

MQCIH structure 63

MQDH structure 102

MQDLH structure 110

MQEPH structure 122

MQIIH structure 166

MQMD structure 197

MQMDE structure 243

MQRFH structure 304

MQRFH2 structure 309

MQRMH structure 320

MQWIH structure 379

formats built-in 197

Function field 63

functions – C programming language 20

G

get-message options structure 126

GetMsgOpts parameter 468

MQCB call 412

MQCB_FUNCTION call 421

GetWaitInterval field 63

GroupId field

MQMD structure 202

MQMDE structure 243

MQPMR structure 302

GroupStatus field 127

H

handle scope 438

handle sharing 86

handles 599

HardenGetBackout attribute 630

Hconn field 688

Hconn parameter

MQBACK call 398, 501

Hconn parameter (*continued*)
 MQBEGIN call 402
 MQBUFMH call 405
 MQCB call 410
 MQCB_FUNCTION call 420
 MQCLOSE call 423
 MQCMIT call 432
 MQCONN call 438
 MQCONNX call 444
 MQCRTMH call 447
 MQCTL call 451
 MQDISC call 458
 MQDLTMH call 461
 MQDLTMP call 464
 MQGET call 467
 MQINQ call 481
 MQINQMP call 495
 MQOPEN call 506
 MQPUT call 524
 MQPUT1 call 539
 MQSET call 550
 MQSETMP call 557
 MQSTAT call 561
 MQSUB call 564
 MQSUBRQ call 571
 MQXCNVC call 691
 scope 438

header files

C programming language 20

Hmsg parameter
 MQCRTMH call 448
 MQDLTMH call 462
 MQDLTMP call 464
 MQINQMP call 495
 MQSETMP call 557

Hobj field
 MQCBC structure 43

Hobj parameter
 MQCB call 411
 MQCLOSE call 423
 MQGET call 467
 MQINQ call 481
 MQOPEN call 513
 MQPUT call 525
 MQSET call 550

I

IGQPputAuthority attribute 594
 IGQUserId attribute 595
 InBuffer parameter 699
 InBufferLength parameter 698
 IndexType attribute 630
 InhibitEvent attribute 595
 InhibitGet attribute 633
 InhibitPut attribute 634
 InitiationQName attribute 634
 InputItem field 64
 inquire message property options 171
 IntAttrCount parameter
 MQINQ call 489
 MQSET call 551
 IntAttrs parameter
 MQINQ call 489
 MQSET call 551
 intra-group queuing 594, 595
 IntraGroupQueuing attribute 595

InvalidDestCount field
 MQOD structure 253
 MQPMO structure 279
 IPAddressVersion attribute
 queue manager 596

K

KeyRepository field
 MQSCO structure 330
 KeyResetCount field
 MQSCO structure 331
 KnownDestCount field 253, 279

L

language declarations 417, 418
 LDAPPassword field
 MQAIR structure 30
 LDAPUserNameLength field
 MQAIR structure 30
 LDAPUserNameOffset field
 MQAIR structure 30
 LDAPUserNamePtr field
 MQAIR structure 31
 LinkType field 64
 ListenerTimer attribute
 queue manager 596
 LocalEvent attribute 597
 LoggerEvent attribute 597
 LTermOverride field 166
 LU62ARMSuffix attribute
 queue manager 598
 LU62Channels attribute
 queue manager 598
 LUGroupName attribute
 queue manager 597
 LUName attribute
 queue manager 598

M

macros 26
 MatchOptions field 128
 MaxActiveChannels attribute
 queue manager 598
 MaxChannels attribute
 queue manager 598
 MaxHandles attribute 599
 MaxMsgLength attribute
 queue 634
 queue manager 599
 MaxPriority attribute 599
 MaxQDepth attribute 635
 MaxUncommittedMsgs attribute 600
 message descriptor extension
 structure 239
 message descriptor structure 181
 message handle to buffer options 247
 message order 475, 534, 547
 MFSMapName field 166
 MQ_CONNECT_TYPE environment
 variable 84
 MQ_DATA_CONV_EXIT call 697
 MQACT_* values 185
 MQADOPT_* values
 AdoptNewMCACheck attribute 579
 AdoptNewMCAType attribute 579
 MQAIR structure 29
 MQAIR_* values 31
 MQAIR_DEFAULT 32
 MQAT_* values
 ApplType
 attribute 655
 field 369
 PutApplType field 217
 MQBACK call 397
 MQBEGIN call 401
 MQBMHO structure 33
 MQBMHO_* values 34
 MQBMHO_DEFAULT 35
 MQBND_* values 625
 MQBO structure 36
 MQBO_* values 36
 MQBO_DEFAULT 37
 MQBOOL 3
 MQBUFMH call 405
 MQBYTE 3
 MQBYTEn 4
 MQCA_* values 482, 551
 MQCB call 409
 MQCB_FUNCTION call 419
 MQCBC structure 38
 MQCBC_* values 44
 MQCBC_DEFAULT 45
 MQCBD structure 46
 MQCBD_* values 50, 51
 MQCBD_DEFAULT 52
 MQCC_* values 661
 MQCCSI_* values 187
 MQCD_CLIENT_CONN_DEFAULT 81
 MQCD_DEFAULT 81
 MQCFUNC_* values 63
 MQCGWI_* values 63
 MQCHAR 4
 MQCHARn 4
 MQCHARV structure 53
 MQCI_* values 189
 MQCIH structure 57
 MQCIH_* values 67
 MQCIH_DEFAULT 70
 MQCLOSE call 423
 MQCLT_* values 64
 MQCLWL_* values 623
 CLWLUseQ attribute 587
 MQCMDL_* values 588
 MQCMHO structure 74
 MQCMHO_DEFAULT 77
 MQCMIT call 431
 MQCNO structure 77
 MQCNO_* values 83, 89
 MQCNO_Accounting_* values 82
 MQCNO_DEFAULT 90
 MQCNOCD structure 80
 MQCO_* values 424
 MQCODL_* values 64
 MQCONN call 435
 MQCONNX call 444
 MQCONNXAny call 80, 446
 MQCRC_* values 66
 MQCRTMH call 447
 MQCSP structure 92

MQCSP_* values 94
MQCSP_DEFAULT 95
MQCT_* values 82
MQCTL call 450
MQCTLO structure 96
MQCTLO_* values 97, 98
MQCTLO_DEFAULT 99
MQCUOWC_* values 68
MQDCC_* values 692
MQDH structure 99
MQDH_* values 103
MQDH_DEFAULT 105, 124
MQDHF_* values 102
MQDISC call 457
MQDL_* values 593, 629
MQDLH structure 106
MQDLH_* values 112, 113
MQDLH_DEFAULT 113
MQDLTMH call 461
MQDLTMP call 464
MQDMHO structure 116
MQDMHO_DEFAULT 117
MQDMPO structure 118
MQDMPO_* values 119
MQDMPO_DEFAULT 120
MQDNSWLM_* values
 DNSWLM attribute 593
MQDPMO_* values 118
MQDXP structure 684
MQDXP_* values 689
MQEC_* values 158
MQEI_* values 192
MQENC_* values 190
MQEPH structure 121
MQEPH_* values 123
MQEVR_* values
 AuthorityEvent attribute 580
 BridgeEvent attribute 580
 ChannelAutoDefEvent attribute 581
 ChannelEvent attribute 581
 CommandEvent attribute 587
 InhibitEvent attribute 595
 LocalEvent attribute 597
 LoggerEvent attribute 597
 PerformanceEvent attribute 602
 QDepthHighEvent attribute 639
 QDepthLowEvent attribute 640
 QDepthMaxEvent attribute 641
 RemoteEvent attribute 610
 SSEvent attribute 611
 StartStopEvent attribute 612
MQEXPI_* values 594
MQFB_* values 111, 193
MQFLOAT32 4
MQFLOAT64 5
MQFMT_* values 197
MQGET call 467
MQGETAny call 478
MQGI_* values 203
MQGMO structure 126
MQGMO_* values 132, 159
MQGMO_DEFAULT 161
MQGS_* values 127
MQHC_* values 458
MQHCONFIG 5
MQHCONN 5
MQHO_* values 423
MQHOBJ 5
MQIA_* values 482, 551
MQIAccounting attribute
 queue manager 600
MQIAUT_* values 165
MQIAV_* values 489
MQICM_* values 165
MQIGQ_* values 595
MQIGQPA_* values 594
MQIIH structure 163
MQIIH_* values 167
MQIIH_DEFAULT 169
MQIMPO structure 171
MQIMPO_* values 178
MQIMPO_DEFAULT 179
MQINQ call 480
MQINQMP call 495
MQINT16 5
MQINT32 5
MQINT64 6
MQINT8 5
MQIPMO_* values 172
MQISS_* values 167
MQIStatistics attribute
 queue manager 601
MQIT_* values 631
MQITIL_* values 167
MQITS_* values 168
MQLONG 6
MQMD structure 181
MQMD_* values 232, 234
MQMD_DEFAULT 236
MQMDE structure 239
MQMDE_* values 244
MQMDE_DEFAULT 245
MQMDEF_* values 243
MQMDS_* values 636
MQMF_* values 204
MQMHBO structure 247
MQMHBO_* values 247, 248
MQMHBO_DEFAULT 248
MQMHBUF call 501
MQMI_* values 210
MQMO_* values 128
MQMON_* values
 MQIStatistics attribute 601
MQMON_* values
 AccountingConnOverride
 attribute 578
 ChannelMonitoring attribute 582
 ChannelStatistics attribute 583
 ClusterSenderMonitoringDefault
 attribute 585
 ClusterSenderStatistics attribute 585
 MQIAccounting attribute 600
 QueueAccounting attribute 607
 QueueMonitoring attribute 644
 QueueStatistics attribute 608
MQMT_* values 211
MQNC_* values 652
MQNPM_* values 637
MQNT_* values 653
MQOD 259
MQOD structure 249
MQOD_* values 260
MQOD_DEFAULT 262
MQOIL_* values 320
MQOL_* values 213
MQOO_* values 507, 626
MQOPEN call 505
MQOR structure 266
MQOR_DEFAULT 267
MQOT_* values 257
 MQSTS structure 363
MQPD 708
MQPER_* values 213
MQPID 6
MQPL_* values 602
MQPMO structure 272
MQPMO_* values 280, 295
MQPMO_DEFAULT 297
MQPMR structure 300
MQPMRF_* values 103, 291
MQPRI_* values 214
MQPSNPRES_* values
 PSNPRES attribute 604
MQPTR 6
MQPUT call 524
MQPUT1 call 539
MQPUT1Any call 548
MQPUTAny call 537
MQQA_* values
 InhibitGet attribute 633
 InhibitPut attribute 634
 Shareability attribute 648
MQQDT_* values 625
MQQSGD_* values 643, 653, 658
MQQSIE_* values 642
MQQT_* values 621, 645
MQRC_* values 196
MQRCVTIME_* values
 ReceiveTimeoutType attribute 609
MQRFH structure 303
MQRFH_* values 305, 313
MQRFH_DEFAULT 306
MQRFH2 708
MQRFH2 structure 308
MQRFH2_DEFAULT 314
MQRL_* values 157
MQRMH structure 315
MQRMH_* values 321, 322
MQRMH_DEFAULT 323
MQRMHF_* values 320
MQRO_* values 222
MQROUTE_* values
 TraceRouteRecording attribute 614
MQRR structure 325
MQRR_DEFAULT 327
MQSCO structure 327
MQSCO_* values 331, 647
MQSCO_DEFAULT 332
MQSCYC_* values
 ScyCase attribute 610
MQSD_* values 348
MQSD_DEFAULT 352
MQSEG_* values 157
MQSET call 550
MQSETMP call 556
MQSID_* values 251
MQSIDT_* values 251
MQSMPO structure 355
MQSMPO_DEFAULT 358
MQSP_* values 613
MQSRO_* values 360

MQSRO_DEFAULT 360
 MQSS_* values 158
 MQSSL_* values
 SSLFIPSRequired attribute 611
 MQSSL_FIPS_* values 330
 MQSTAT call 561
 MQSTS structure 361
 MQSTS_* values 364
 MQSUB call 564
 MQSUBRQ call 571
 MQSVC_* values
 ChannelInitiatorControl attribute 582
 MQTC_* values 648
 MQTCPKEEP_* values
 TCPKeepAlive attribute 613
 MQTCPSTACK_* values
 TCPStackType attribute 614
 MQTID 6
 MQTM structure 366
 MQTM_* values 371
 MQTM_DEFAULT 372
 MQTMC_* values 375
 MQTMC2 structure 374
 MQTMC2_DEFAULT 376
 MQTRAXSTR_* values
 ChinitTraceAutoStart attribute 584
 MQTT_* values 650
 MQUINT16 6
 MQUINT32 6
 MQUINT64 7
 MQUINT8 6
 MQULONG 7
 MQUS_* values 650
 MQWL_* values 160
 MQWIH structure 378
 MQWIH_* values 380
 MQWIH_DEFAULT 381
 MQXC_* values 383
 MQXCC_* values 385
 MQXCNVC call 691
 MQXDR_* values 687
 MQXP structure 383
 MQXP_* values 385, 386
 MQXQH structure 387
 MQXQH_* values 391
 MQXQH_DEFAULT 392
 MQXR_* values 384
 MQXT_* values 384
 MQXUA_* values 385
 MsgDeliverySequence attribute 636
 MsgDesc field 390
 MsgDesc parameter
 MQ_DATA_CONV_EXIT call 698
 MQCB call 411
 MQCB_FUNCTION call 421
 MQGET call 467
 MQPUT call 525
 MQPUT1 call 540
 MsgFlags field
 MQMD structure 204
 MQMDE structure 243
 MsgHandle field
 MQGMO structure 130
 MsgId field
 MQMD structure 208
 MQPMR structure 302
 MsgMarkBrowseInterval attribute 601

MsgSeqNumber field
 MQMD structure 210
 MQMDE structure 243
 MsgToken field 131, 379
 MsgType field 211

N
 Name parameter
 MQSETMP call 557
 NameCount attribute 652
 named constants – COBOL programming language 25
 namelist attributes 651
 NamelistDesc attribute 652
 NamelistName attribute 653
 NamelistType attribute 653
 Names attribute 653
 NameValueCCSID field 309
 NameValueData field 310
 NameValueLength field 312
 NameValueString field 304
 NextTransactionId field 64
 NonPersistentMessageClass attribute 637
 notational conventions
 C programming language 23
 COBOL programming language 25
 S/370 assembler programming language 29

O
 ObjDesc parameter
 MQOPEN call 506
 MQPUT1 call 539
 object descriptor structure 249
 object record structure 266
 ObjectInstanceId field 320
 ObjectName field
 MQOD structure 253
 MQOR structure 266
 MQSTS structure 362
 ObjectQMgrName field
 MQOD structure 254
 MQOR structure 266
 MQSTS structure 363
 ObjectRecOffset field
 MQDH structure 102
 MQOD structure 255
 ObjectRecPtr field 256
 ObjectType field
 MQOD structure 257
 MQRMH structure 320
 MQSTS structure 363
 Offset field
 MQMD structure 212
 MQMDE structure 243
 OpenInputCount attribute 637
 OpenOutputCount attribute 637
 Operation parameter
 MQCB call 410
 MQCTL call 451
 Options field
 MQBMHO structure 34
 MQBO structure 36

Options field (*continued*)
 MQCBD structure 50
 MQCMHO structure 75
 MQCNO structure 82
 MQCTLO structure 97
 MQDMHO structure 116
 MQDPMO structure 118
 MQGMO structure 132
 MQIPMO structure 172
 MQMHBO structure 247
 MQPMO structure 280
 MQSMPO structure 356
 Options parameter
 MQCLOSE call 423
 MQOPEN call 506
 MQXCNVC call 691
 ordering of messages 475, 534, 547
 OriginalLength field
 MQMD structure 212
 MQMDE structure 243
 OriginalMsgHandle field
 MQPMO structure 279, 290
 OutboundPortMax attribute
 queue manager 602
 OutboundPortMin attribute
 queue manager 602
 OutBuffer parameter 699
 OutBufferLength parameter 699
 OutputDataLength field 64

P
 parameters with undefined data types 20
 PCFHeader field
 MQEPH structure 122
 PerformanceEvent attribute 602
 persistence 627
 Persistence field 213
 Platform attribute 602
 PMQACH 7
 PMQAIR 7
 PMQAXC 7
 PMQAXP 7
 PMQBMHO 7
 PMQBO 7
 PMQBOOL 7
 PMQBYTE 7
 PMQBYTEn 7
 PMQCBC 7
 PMQCBD 7
 PMQCHAR 7
 PMQCHARn 8
 PMQCHARV 8
 PMQCIH 8
 PMQCMHO 8
 PMQCNO 8
 PMQCSP 8
 PMQCTLO 8
 PMQDH 8
 PMQDHO 8
 PMQDLH 8
 PMQDMHO 8
 PMQDMPO 8
 PMQEPH 8
 PMQFLOAT32 8
 PMQFLOAT64 8

PMQFUNC 8
 PMQGMO 8
 PMQHCONFIG 8
 PMQHCONN 9
 PMQHMSG 9
 PMQHOBJ 9
 PMQIIH 9
 PMQIMPO 9
 PMQINT16 9
 PMQINT32 9
 PMQINT64 9
 PMQINT8 9
 PMQLONG 9
 PMQMD 9, 10
 PMQMDE 9
 PMQMDI 9
 PMQMHB0 9
 PMQOD 9
 PMQOR 9
 PMQPID 9, 10
 PMQPMO 10
 PMQPTR 10
 PMQRFH 10
 PMQRFH2 10
 PMQRMH 10
 PMQRR 10
 PMQSCO 10
 PMQSD 10
 PMQSMPO 10
 PMQSRO 10
 PMQSTS 10
 PMQTID 10
 PMQTM 10
 PMQTM2 10
 PMQUINT16 10
 PMQUINT32 10
 PMQUINT64 11
 PMQUINT8 10
 PMQULONG 11
 PMQVOID 11, 397
 PMQWIH 11
 PMQXQH 11
 pointer data type – COBOL programming
 language 25
 Priority field 214
 process definition attributes 654
 ProcessDesc attribute 657
 ProcessName
 attribute
 process definition 658
 queue 638
 field
 MQTM structure 370
 MQTMC2 structure 375
 PropDesc parameter
 MQSETMP call 557
 property descriptor structure 417, 418
 PropertyControl 603, 638
 PubSubMaxMsgRetryCount attribute
 queue manager 605
 PubSubMode attribute
 queue manager 605
 PubSubNPInputMsg attribute
 queue manager 604
 PubSubNPResponse attribute
 queue manager 604

PubSubSyncPoint attribute
 queue manager 605
 put message record structure 300
 put-message options structure 272
 PutAppName field
 MQDLH structure 110
 MQMD structure 216
 PutAppType field
 MQDLH structure 110
 MQMD structure 217
 PutDate field
 MQDLH structure 110
 MQMD structure 219
 PutFailureCount field 362
 PutMsgOpts parameter
 MQPUT call 525
 MQPUT1 call 540
 PutMsgRecFields field
 MQDH structure 103
 MQPMO structure 291
 PutMsgRecOffset field
 MQDH structure 103
 MQPMO structure 292
 PutMsgRecPtr field 293
 PutSuccessCount field 364
 PutTime field
 MQDLH structure 111
 MQMD structure 219
 PutWarningCount field 364

Q

QDepthHighEvent attribute 639
 QDepthHighLimit attribute 639
 QDepthLowEvent attribute 640
 QDepthLowLimit attribute 640
 QDepthMaxEvent attribute 641
 QDesc attribute 641
 QMgrDesc attribute 606
 QMgrIdentifier attribute 606
 QMgrName
 attribute 606
 field 375
 QMgrName parameter
 MQCONN call 436
 MQCONN call 444
 QName
 attribute 642
 field
 MQTM structure 371
 MQTMC2 structure 375
 QServiceInterval attribute 642
 QServiceIntervalEvent attribute 642
 QSGDisp attribute
 namelist 653, 658
 queue 643
 QSGName attribute 607
 QType attribute 645
 queue attributes 615
 queue manager attributes 575
 queue-manager aliasing 616
 queue-sharing group 255, 436, 607
 queue, dynamic 506
 QueueAccounting attribute 644
 queue manager 607
 QueueMonitoring attribute
 queue 644

QueueMonitoring attribute (*continued*)
 queue manager 607
 QueueStatistics attribute 645
 queue manager 608

R

reason codes
 alphabetic list 661
 Reason field 65
 MQCBC structure 43
 MQDLH structure 111
 MQDXP structure 688
 MQRR structure 326
 MQSTS structure 363
 Reason parameter
 MQBACK call 398, 449
 MQBEGIN call 402
 MQCB call 413
 MQCLOSE call 427
 MQCMIT call 432
 MQCONN call 439
 MQCONN call 445
 MQCTL call 453
 MQDISC call 458
 MQDLTMH call 462
 MQDLTMP call 465
 MQGET call 469
 MQINQ call 490
 MQINQMP call 498
 MQMHBUFF call 407, 503
 MQOPEN call 513
 MQPUT call 527
 MQPUT1 call 541
 MQSET call 552
 MQSETMP call 559
 MQSTAT call 562
 MQSUB call 567, 572
 MQXCNCV call 695
 ReceiveTimeout attribute
 queue manager 608
 ReceiveTimeoutMin attribute
 queue manager 609
 ReceiveTimeoutType attribute
 queue manager 609
 RecsPresent field
 MQDH structure 103
 MQOD structure 257
 MQPMO structure 293
 reference message header structure 315
 RemoteEvent attribute 610
 RemoteQMGrName
 attribute 645
 field 391
 RemoteQName
 attribute 646
 field 391
 RemoteSysId field 65
 RemoteTransId field 65
 reply queue aliasing 616
 ReplyToFormat field 65, 166
 ReplyToQ field 220
 ReplyToQMGr field 221
 Report field
 MQMD structure 221
 using 671
 report message conversion 683

- RepositoryName attribute 610
- RepositoryNameList attribute 610
- RequestedCCSID field
 - MQIPMO structure 177
- RequestedEncoding field
 - MQIPMO structure 177
- Reserved field 98, 177
 - MQIIH structure 166
 - MQWIH structure 380
 - MQXP structure 385
- Reserved1 field 65, 156
 - MQCSP structure 94
 - MQPMO structure 156
- Reserved2 field 65
 - MQCSP structure 94
- Reserved3 field 65
- Reserved4 field 66
- ResolvedObjectName field
 - MQOD structure 363
- ResolvedQMgrName field
 - MQOD structure 258
 - MQPMO structure 293
 - MQSTS structure 363
- ResolvedQName field
 - MQGMO structure 157
 - MQOD structure 258
 - MQPMO structure 294
- ResolvedType 259
- response record structure 325
- ResponseRecOffset field
 - MQOD structure 259
 - MQPMO structure 294
- ResponseRecPtr field
 - MQOD structure 260
 - MQPMO structure 295
- RetentionInterval attribute 646
- return codes 661
- ReturnCode field 66
- ReturnedCCSID field
 - MQIPMO structure 177
- ReturnedEncoding field
 - MQIPMO structure 177
- ReturnedLength field 157
- ReturnedName field
 - MQIPMO structure 177
- rules and formatting header
 - structure 303
- rules and formatting header structure
 - version 2 308

S

- Scope attribute 647
- scope, handles 438
- ScyCase attribute 610
- security parameters 92
- SecurityParmsOffset field
 - MQCNO structure 88
- SecurityParmsPtr field
 - MQCNO structure 88
- SecurityScope field 167
- Segmentation field 157
- SegmentStatus field 158
- SelectorCount parameter
 - MQINQ call 481
 - MQSET call 550

- Selectors parameter
 - MQINQ call 481
 - MQSET call 550
- ServiceName field 380
- ServiceStep field 380
- set message property options
 - structure 355
- Shareability attribute 648
- shared handles 86
- shared queue 255, 474
- SharedQMgrName attribute
 - queue manager 611
- sharing conversations 86, 87
- Signal1 field 158
- Signal2 field 159
- SourceBuffer parameter 694
- SourceCCSID parameter 694
- SourceLength parameter 694
- SrcEnvLength field 320
- SrcEnvOffset field 321
- SrcNameLength field 321
- SrcNameOffset field 321
- SSL configuration options structure 327
- SSLConfigOffset field 88
- SSLConfigPtr field 88
- SSLEvent attribute
 - queue manager 611
- SSLFIPSRequired attribute
 - queue manager 611
- SSLKeyResetCount attribute
 - queue manager 331, 612
- StartCode field 66
- StartStopEvent attribute 612
- Stat parameter
 - MQSTAT call 562
- StatisticsInterval attribute
 - queue manager 613
- Status structure 361
- StorageClass attribute 648
- StrucId field
 - MQAIR structure 31
 - MQBMHO structure 34
 - MQBO structure 36
 - MQCBC structure 44
 - MQCBD structure 51
 - MQCIH structure 67
 - MQCMHO structure 76
 - MQCNO structure 89
 - MQCSP structure 94
 - MQCTLO structure 98
 - MQDH structure 103
 - MQDLH structure 112
 - MQDMHO structure 116
 - MQDMPO structure 119
 - MQDXP structure 689
 - MQEPH structure 123
 - MQGMO structure 159
 - MQIIH structure 167
 - MQIMPO structure 178
 - MQMD structure 232
 - MQMDE structure 244
 - MQMHBO structure 247
 - MQOD structure 260
 - MQPMO structure 295
 - MQRFH structure 305
 - MQRFH2 structure 313
 - MQRMH structure 321

- StrucId field (*continued*)
 - MQSCO structure 331
 - MQSD structure 348
 - MQSMPO structure 357
 - MQSRO structure 360
 - MQSTS structure 364
 - MQTM structure 371
 - MQTMC2 structure 375
 - MQWIH structure 380
 - MQXP structure 385
 - MQXQH structure 391
- StrucLength field
 - MQCIH structure 67
 - MQDH structure 104
 - MQEPH structure 123
 - MQIIH structure 167
 - MQMDE structure 244
 - MQRFH structure 305
 - MQRFH2 structure 313
 - MQRMH structure 322
 - MQWIH structure 380
- structures – COBOL programming
 - language 24
- SubDesc parameter
 - MQSUB call 564
- syncpoint 613
- SyncPoint attribute 613
- System/390 Assembler programming
 - language
 - macros 26
 - notational conventions 29

T

- TargetBuffer parameter 695
- TargetCCSID parameter 695
- TargetLength parameter 695
- TaskEndStatus field 67
- TCPChannels attribute
 - queue manager 613
- TCPKeepAlive attribute
 - queue manager 613
- TCPName attribute
 - queue manager 614
- TCPStackType attribute
 - queue manager 614
- Timeout field 295
- TraceRouteRecording attribute
 - queue manager 614
- TranInstanceId field 167
- TransactionId field 68
- transmission queue header structure 387
- TranState field 168
- trigger message structure 366
- TriggerControl attribute 648
- TriggerData
 - attribute 649
 - field
 - MQTM structure 371
 - MQTMC2 structure 375
- TriggerDepth attribute 649
- triggering 648
- TriggerInterval attribute 614, 615
- TriggerMsgPriority attribute 649
- TriggerType attribute 650
- trusted application 83

Type parameter
 MQINQMP call 496
 MQSETMP call 557
 MQSTAT call 562
TypeString field
 MQIPMO structure 178

U

UCS-2 757
Uncommitted messages 600
Unicode 757
UnknownDestCount field
 MQOD structure 260
 MQPMO structure 295
UOWControl field 68
Usage attribute 650
use from C++ 22
UserData
 attribute 658
 field
 MQTM structure 371
 MQTMC2 structure 375
UserIdentifier field 233
UTF-16 757
UTF-8 757

V

Value parameter
 MQINQMP call 497
 MQSETMP call 558
ValueCCSID field
 MQSMPO structure 357
ValueEncoding field
 MQSMPO structure 357
ValueLength parameter
 MQINQMP call 497
 MQSETMP call 558
variable length string structure 53
Version field
 MQAIR structure 31
 MQBMHO structure 34
 MQBO structure 36
 MQCBC structure 44
 MQCBD structure 51
 MQCIH structure 69
 MQCMHO structure 76
 MQCNO structure 89
 MQCSP structure 94
 MQCTLO structure 98
 MQDH structure 104
 MQDLH structure 113
 MQDMHO structure 116
 MQDMPO structure 119
 MQDXP structure 689
 MQEPH structure 123
 MQGMO structure 159
 MQIIH structure 168
 MQIMPO structure 178
 MQMD structure 234
 MQMDE structure 244
 MQMHBO structure 248
 MQOD structure 261
 MQPMO structure 296
 MQRFH structure 306

Version field (*continued*)
 MQRFH2 structure 313
 MQRMH structure 322
 MQSCO structure 331
 MQSD structure 350
 MQSMPO structure 357
 MQSRO structure 360
 MQSTS structure 364
 MQTM structure 371
 MQTMC2 structure 375
 MQWIH structure 380
 MQXP structure 386
 MQXQH structure 391
VSBufSize field 54
VSLength field 54
VSOffset field 55
VSPtr field 55

W

WaitInterval field 160

X

XmitQName attribute 651

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To make comments about the functions of IBM products or systems, talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

User Technologies Department (MP095)
IBM United Kingdom Laboratories
Hursley Park
WINCHESTER,
Hampshire
SO21 2JN
United Kingdom

- By fax:
 - From outside the U.K., after your international access code use 44-1962-816151
 - From within the U.K., use 01962-816151
- Electronically, use the appropriate network ID:
 - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
 - IBMLink™: HURSLEY(IDRCF)
 - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.



SC34-6940-01



Spine information:



WebSphere MQ

Application Programming Reference

Version 7.0